DOCUMENT RESUME

ED 143 507 SE 022 984

AUTHOR Charp, Sylvia; And Others
TITLE Algorithms, Computation and Mathematics. Teacher's Commentary. Revised Edition.
INSTITUTION Stanford Univ., Calif. School Mathematics Study Group.
SPONS AGENCY National Science Foundation, Washington, D.C.
PUB DATE 66
NOTE 302p.; For related documents, see SE 022 983-988; Not available in hard copy due to marginal legibility of original document

EDRS PRICE MF-$0.83 Plus Postage. HC Not Available from EDRS.
DESCRIPTORS *Algorithms; *Computers; *Instructional Materials; Mathematics Education; Programing Languages; Secondary Education; *Secondary School Mathematics; *Teaching Guides
IDENTIFIERS *School Mathematics Study Group

ABSTRACT

The materials in this teacher's guide are designed for about 18 weeks of study by secondary school students. For maximum benefit, the student needs contact with a computer, primarily for verifying and trouble-shooting the algorithms which he or she has constructed. The course is usually taught for grade 11 or 12 students. The commentary contains background material, suggestions for use, and answers for exercises for each chapter of the student text. Comments indicate the course requires more preparation time for the teacher than most high school mathematics courses; use of a student assistant is recommended. (RH)

ALGORITHMS, COMPUTATION AND MATHEMATICS

Teacher's Commentary

Revised Edition

The following is a list of all those who participated in the preparation of this volume:

Sylvia Charp, Dobbins Technical High School, Philadelphia, Pennsylvania
Alexandra Forsythe, Gunn High School, Palo Alto, California
Bernard A Galler, University of Michigan, Ann Arbor, Michigan
John G. Herriot, Stanford University, California
Walter Hoffmann, Wayne State University, Detroit, Michigan
Thomas E. Hull, University of Toronto, Toronto, Ontario, Canada
Thomas A. Keenan, University of Rochester, Rochester, New York
Robert E. Monroe, Wayne State University, Detroit, Michigan
Silvio O. Navarro, University of Kentucky, Lexington, Kentucky
Elliott I. Organick, University of Houston, Houston, Texas
Jesse Peckenham, Oakland Unified School District, Oakland, California
George A. Robinson, Argonne National Laboratory, Argonne, Illinois
Phillip M. Sherman, Bell Telephone Laboratories, Murray Hill, New Jersey
Robert E. Smith, Control Data Corporation, St. Paul, Minnesota
Warren Stenborg, University of Minnesota, Minneapolis, Minnesota
Harley Tillitt, U. S. Naval Ordnance Test Station, China Lake, California
Lyneve Waldrop, Newton South High School, Newton, Massachusetts

The following were the principal consultants:

George E. Forsythe, Stanford University, California
Bernard A. Galler, University of Michigan, Ann Arbor, Michigan
Wallace Givens, Argonne National Laboratory, Argonne, Illinois

Permission to make verbatim use of material in this book must be secured from the Director of SMSG. Such permission will be granted except in unusual circumstances. Publications incorporating SMSG materials must include both an acknowledgment of the SMSG copyright (Yale University or Stanford University, as the case may be) and a disclaimer of SMSG endorsement. Exclusive license will not be granted save in exceptional circumstances, and then only by specific action of the Advisory Board of SMSG.

Financial support for the School Mathematics Study Group has been provided by the National Science Foundation.

TABLE OF CONTENTS

PREFACE TO THE TEACHER

Weekly Teaching Plan

You will find below a rough guide for the number of weeks which each chapter is expected to take for adequate coverage. Here weeks are assumed to be full five day weeks without interruption. These estimates are based on the experience gained by the seventeen teachers who used the book during the trial year. The times shown include those needed to cover the corresponding material in the language supplements.

Chapter	Min.	Max.	Best Guess
1	$\frac{1}{2}$	1	1
2	2	3	$2\frac{1}{2}$
3	2	3	$2\frac{1}{2}$
4	2	3	2
5	$2\frac{1}{2}$	3	$2\frac{1}{2}$
6	1	1	1
7	2	3	3
8	3	4	4
Epilogue	0	0	0
Subtotals	15	21	$17\frac{1}{2}$
Appendix A	1	1	1
Appendix B (optional)			

Assuming 15 weeks of uninterrupted instruction, it would appear that either Chapter 7 or Chapter 8 might be covered with moderate thoroughness (but not both).

The Rôle of Laboratory Work in this Course

For maximum benefit from this course, the student needs contact with a computer, primarily for verifying and trouble-shooting the algorithms which he has constructed. This laboratory work need not be "hands-on" computer experience, but should be whatever is necessary for adequate testing of programs on the computer facility that is chosen for use. Hands-on computer experience is no goal of this course. Where possible, it should be avoided as there is more than enough substantial material of a mathematical nature in this text.

Many schools already have computers or have them on order. For schools that have no facilities of their own, (and even for schools that do), arrangements with nearby colleges, universities, or other institutions for the use of their facilities are urged. Distances of up to 200 miles from the university or college need not be a prohibitive factor. A number of universities will be installing time-shared computer systems in the coming years. Arrangements for remote use of such facilities (especially in a city) via rented teletype or other keyboard consoles looks increasingly promising.

Choice of a Programming Language

Careful and serious study must be devoted to the choice of a language, a computer, and a software system. (Software refers to the service programs used to operate and exploit the computer for your benefit.) The best language is not necessarily measured by the number of people using it.

Learning one good programming language makes it very easy to learn another. If the flow chart language in this book is taught well, then ALGOL, FORTRAN, MAD, or any similar language can be learned with comparative ease as the second language. The flow chart language attempts to deal with those concepts of central interest in all programming languages in a context free of most of the syntactic details associated with a given language.

Given a choice of one programming language, such as FORTRAN, one must still be aware of variations in its implementation on various types of computers. Sometimes even two versions of FORTRAN for the same machine may differ materially.

The device which implements a language on a given machine is a program which we call a "processor." Among the factors that contribute to variation in these processors are

(a) size of the computer (memory and speed);

(b) processing speeds--rate at which programs are compiled (or readied by the computer for computation);

(c) speed in executing the computation;

(d) processing quality--some processors in use still have errors in them. Some processors produce better intermediate documentation than others;

(e) error detection and recovery--a critical factor is the degree to which programming and language errors made by the student or teacher are detected, clearly identified, and reported or corrected by the computer at the time the program is being compiled. Fatal programming errors can also be detected and reported during execution of the target (compiled) program. The quality of this reporting also varies widely depending on the executive system (more software) which is being used. Detection of equipment malfunction during input/output and of misuse of library subroutines is important, and the way these errors are treated can be critical to the efficient running of large numbers of small student problems on a computer in a reasonable amount of elapsed time.

Next in importance to the quality of the programming language is the question of access to the computer and the nature of the response by the computer. Here one must often choose between hands-on experience and remote use. Each technique depends for good results on the response which is obtained from the computer system. Response is measured by

(a) "turn-around" time;

(b) amount and pertinence of information received, especially during error detection and reporting or recovery.

A computer 100 miles away that is equipped with software ideal for school work may prove to be superior to a small, inadequately supported "hands-on" computer in the next room. The teacher who, by choice or necessity, uses a small home-owned computer must be cognizant of the trade-off between technical training and machine-oriented know-how which tends to accrue from hands-on use versus the facility that is gained in focusing on programming language, algorithmic construction, mathematical theory and more realistic insight into the use of computers in science, industry, and business that can be gained from working with a better software system on a remote machine.

The Computer Language Supplement

In order to increase the scope of applicability of this book, the specific syntactic details of the computer language have been split off from the main flow chart text into a language supplement. This enables the school to choose between FORTRAN, ALGOL, MAD or any other as the computer language, while still preserving compatibility concerning the fundamental concepts that are common to most computer languages. Each supplement is then briefer and easier to revise as the languages change with time. This way of organizing the material also makes it possible to write other language supplements for new programming languages. Only three have been prepared by SMSG at this time, but these may be used as models for others.

A direct consequence of this separation is the necessity to learn another language (the flow chart language) first. This is considered an asset, not a liability. You must, however, be prepared to give the student some additional guidance in correlating his study of the two parallel texts.

After the text has been studied, you may need to examine reference materials that deal with the specific implementations for the language and machine which you have chosen to use. Neither teacher nor student should read a reference manual as a text. If at all possible, you should seek the assistance of capable computer specialists in helping you to identify and interpret the needed information in the reference manual.

There are a number of commercially available primers, guides, and texts that deal with FORTRAN, ALGOL and MAD. Those available at present are not generally compatible with SMSG mathematical concepts or not aimed at a level appropriate to the high school audience. Some of these may be useful to the teacher as sources of problems and illustrations. Better students can be referred to them after acquiring a healthy grasp of the material in the flow chart text and language supplement.

Organization of the language supplements

(a) There is no counterpart to Chapter 1 of the main text.

(b) Chapter 2 should be studied only after completing the counterpart (in its entirety) in the main text. Reason--we want to introduce a (small but for some purposes complete) flow chart language. Then we introduce its programming language equivalent. In this way FORTRAN, ALGOL, MAD or what have you, is introduced as a second language.

(c) Lab Work

Upon completing Chapter 2 of the supplement, small computer programs can be run as laboratory exercises. To do this certain fill-in information must be conveyed by the teacher.

1. How to fill-out coding forms.

2. How to key punch cards or punch paper tape, etc.

3. How to prepare decks of cards including I.D. cards, monitor specification cards, if any.

The teacher is urged to get help from local practitioners on these details. Consult reference manuals only as a last resort.

(d) Subsequent chapters add more language capability. Each of these can be read in conjunction with, i.e., section by section with, the main text.

(e) I/O details

Format details are necessary in some languages like FORTRAN. There is a risk of giving too much detail. To solve this problem, the FORTRAN supplement offers format details piecemeal--as needed beginning with Chapter 2. Many reference manuals are difficult to read on this subject.

Format in a language like MAD need not be taught because a set of "simplified" input-output statements are available which obviates the need to teach format codes and associated details.

Format in a language like ALGOL is not treated because as yet no standard way of handling I/O is part of what is called ALGOL. Computer implementation of the input-output procedures in ALGOL differ. We must rely on the teacher being shown the minimum necessary information by the local practitioner. Hopefully the particular ALGOL implementation uses a very simply I/O scheme involving a minimum of format control or none at all.

Helping students to debug their programs

The literature on this subject has little formal development of wide applicability. There are so many different software systems that we can't give standard procedures. All we can do is give general principles.

(a) Debugging is an iterative process. The loop includes the flow chart, the program, sometimes the data format and the output format and sometimes even the problem statement. Iteration is a process of refinement of all steps in the loop.

(b) To minimize error certain good work habits are essential.

1. Be methodical, neat--keep clear notes on the statement of the problem, assumptions used, and the symbols used and their significance.

2. Draw a flow chart that conveys an up to date version of the algorithm being tested.

3. Keep a check-list of simple things to remember and use it each time before submitting the program to the computer. The check-list consists of a set of simple clerical details and tasks, the violation of any one of which, however trivial, can fault a computer run.

"4. Develop the habit of getting the most useful information from the computer at each run but not too much detail. *Think of each run as a debugging run and insert extra output statements which can later be removed with ease.* These output statements are to print selected values developed at intermediate points in the computation. Suppose we find that a printed intermediate result is demonstrably incorrect. Then we can confine our search for the trouble to that small section of the program which develops this result incorrectly. It may be that, after some study of this section, we still cannot deduce the reason for the erroneous result. We may then insert additional output statements at key points within the suspected section of the program and rerun, hoping finally to pinpoint the incorrect program step.

Pre-Laboratory Planning and Preparation

The teacher has quite a few problems to solve before the class begins its laboratory work. You must decide:

1. How many programs will each student submit per week, month, semester? Make a schedule for students to follow based on *turn around time*.

2. Will each student run all programs or will he run selected programs?

3. What arrangements will you make for card or tape preparation time?

Obviously, you should seek consultation (formal or informal) with university or high school teachers who already have the experience you still need.

Workload for the Teacher

It should now be fairly apparent that teaching ACM (Algorithms, Computation and Mathematics) as a regular high school course involves considerably more work and time than does the teaching of an ordinary course in high school mathematics. Until it becomes commonplace for the high school to have its own staffed computer laboratory, the ACM teacher must take on many of the duties of such a laboratory staff. Many of these activities and responsibilities can be delegated to others, however. Teachers have experimented by having the better students assist with a number of the laboratory chores, and errands. Where especially bright students are available, or where eleventh graders are allowed to enroll, certain twelfth graders can become computer operators, graders or assistants for helping others in program debugging. After teaching ACM several times, and "organizing", it looks like it would be possible to reduce the overload to manageable proportions.

Chapter T1

ALGORITHMS, LANGUAGE, AND MACHINES

Summary of Chapter 1

Section 1-1, Introduction sketches briefly a few reasons for studying about computers and the areas where they are being employed to solve problems. Some students may want to gain a more complete picture of computers and their impact on society. We suggest as outside reading the Section 11, New York Times, Sunday, April 24, 1966, entitled "The Computer and Society, Six Viewpoints".

Section 1-2, History gives a very brief account of some of the developments leading to the stored program digital electronic computer.

In Section 1-3, Some Technical Aspects of Computers, some of the more fundamental ideas connected with the organization of computers and the method of their operation are illustrated with the aid of a hypothetical computer SAMOS. The details of the SAMOS computer are not in themselves sufficiently important to warrant learning for retention but are offered only as a means of illustrating the more basic ideas that the hypothetical machine embodies. On the occasions when specific details about SAMOS are needed, the student can refer to Appendix A.

In Section 1-4, Numbers and Other Characters, we see how numerical data(i.e., the integers and the reals) and alphanumerical data (strings of characters) might be represented or "coded" internally, i.e., in memory. The "floating-point" coding scheme for the reals is introduced. Also, sets of binary codes are suggested for alphabets consisting of both digits and letters. Finally, the memory is viewed as capable of storing arbitrary strings of characters grouped one or more per memory word, depending upon the size or length of the word.

In Section 1-5, Algorithms, we come to the central topic of this text. For this course we are interested in details of actual computers only to the extent that they are necessary to motivate or explain aspects of the study of algorithm construction. Because it is more important, one might hold that this section should precede Section 1-3. However, Section 1-3 provides a concrete basis for beginning the study of algorithms, and, after some soul searching, the present order was selected.

Finally, Section 1-6, Comments on Language, points out the need for an adequate language in which to express algorithms, laying the groundwork for the study of the flow chart language in Chapters 2 through 5.

Chapter 1 is intended primarily as a reading assignment at the beginning of the course to gain some general background knowledge of computers and the concept of algorithms. It is not necessary that the study of this material build motivation for what follows. Experience shows that student interest automatically heightens upon launching into Chapter 2.

You will undoubtedly want to discuss some of the material from Sections 1-3, 1-4, and 1-5 in class. But certainly no more than one week should be spent on the entire chapter. We mention this important point for the obvious reason that this chapter will seem to raise more questions that it answers and lengthy digressions can easily result. This is not a "teaching" chapter as are the chapters which follow. Some students will want to know much more about how computers are built or how they work. You might encourage your better students to read Appendix A on their own. Many questions on how computers function will be answered here. Questions on the construction and organization of computer hardware are considered to be outside the scope of this course. However, a prime reference on this topic for high school students is: "The Man Made World" published 1965 by the Commission on Engineering Education.

Problems

Only one set of problems is given but this set should not be overlooked. These are the four ball weighing problems at the end of Example 2 in Section 1-4. Solutions to these are given in following paragraphs. It would be worthwhile to assign 1 and 2 or 3 and 4 as homework. We would not recommend more problems at this time.

Tests

We suggest that no tests are needed on the material in this chapter. If tests are given, the questions should be confined to material on Sections 1-3, 1-4, and 1-5.

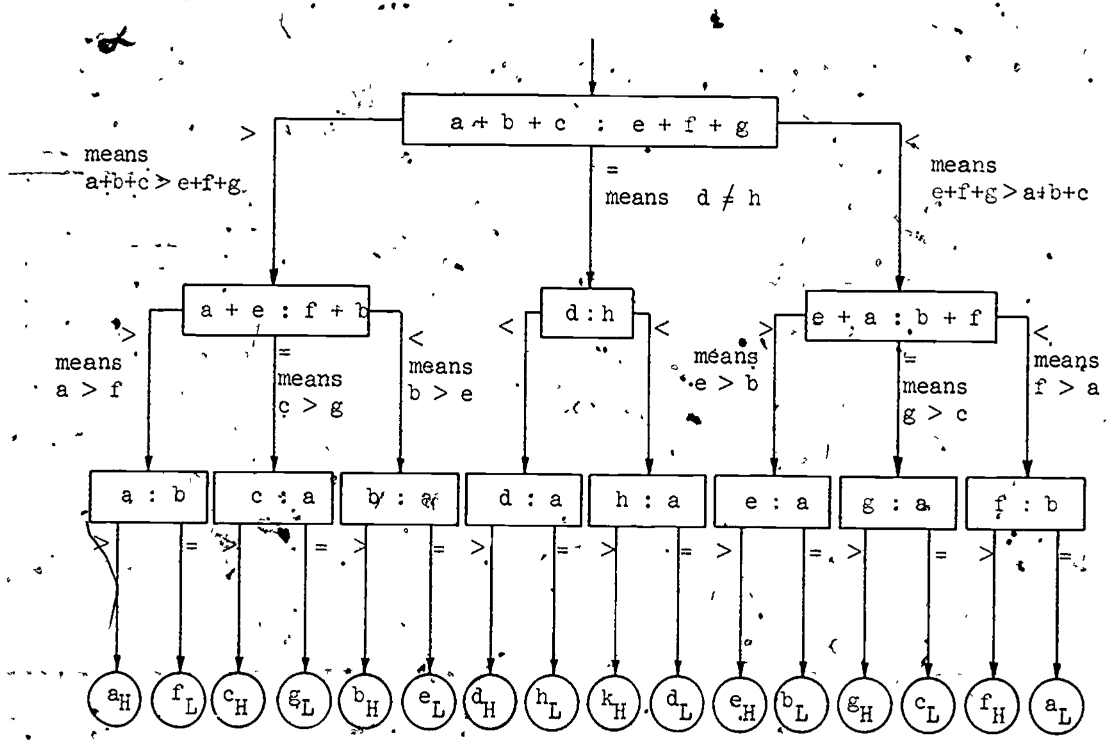
Solutions to the Weighing Problems

Problem 1. Weighing 8 balls.

Two solutions are given:

Solution No. 1

A flow chart approach. Conclusions are subscripted H for heavy and L for light.



Strategy

1. Select and weigh a subset of the balls and consider the significance of the three possible outcomes. If the subset balances, we know that the remaining balls contain the one we are interested in.

2. Once we have isolated a pair containing the "odd" ball and we want to know if one of them is heavy or light, we weigh one of the two candidates against any other which is known to be "standard".

Solution No. 2

Let the balls be A, B, C, D, E, F, G, H. Weigh the balls as follows and note results each time:

1st time	A, B, C, D,	E, F, G, H
2nd time	A, B, G	C, D, E
3rd time	A, C, H	B, D, F

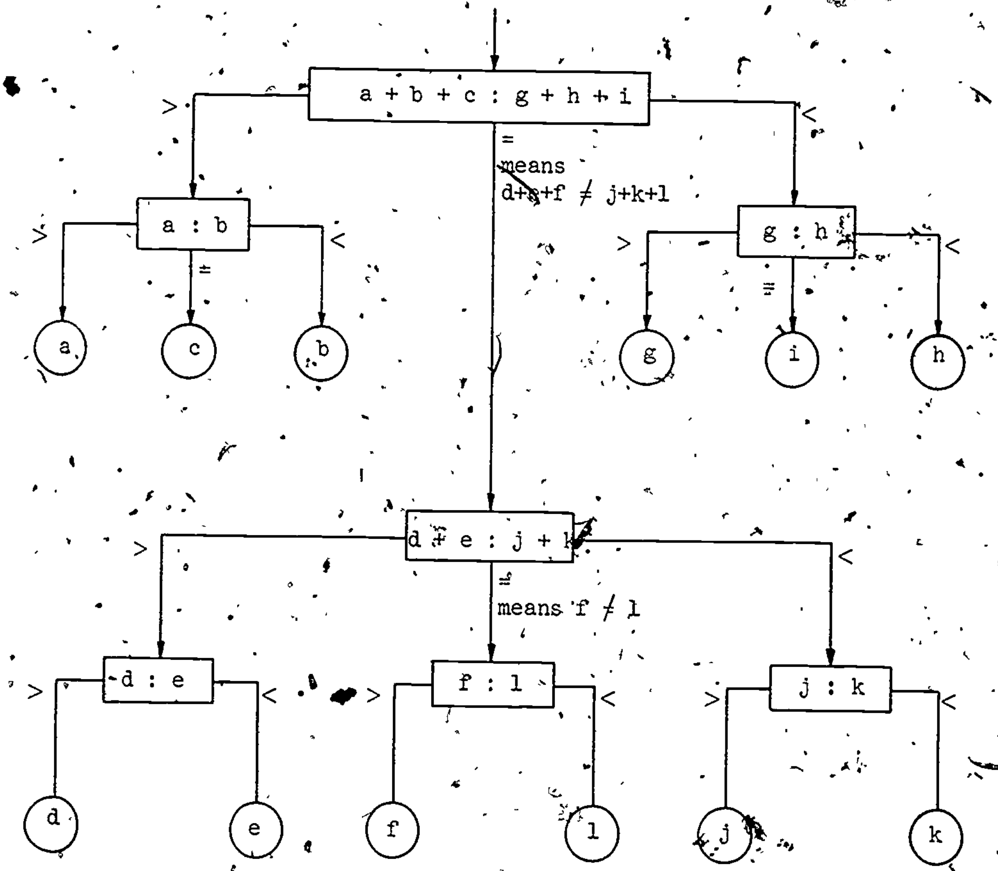
If the left side of a weighing was down, we write L.
If the right side of a weighing was down, we write R.
If the sides balanced, we write "=".

The results of every set of three observations (weighings) can be coded as a "triple" or string of three characters involving "L", "R", or "=". For each of the listed 16 triples which can result from the above weighing sequence, we give the associated (unique) conclusion. (Other weighing sequences might have been chosen.)

Triple	Conclusion
L L L	A is heavy
R R R	A is light
L L R	B is heavy
R R L	B is light
L R L	C is heavy
R L R	C is light
L R R	D is heavy
R L L	D is light
L L =	E is light
R R =	E is heavy
L = L	F is light
R = R	F is heavy
L R =	G is light
R L =	G is heavy
L = R	H is light
R = L	H is heavy

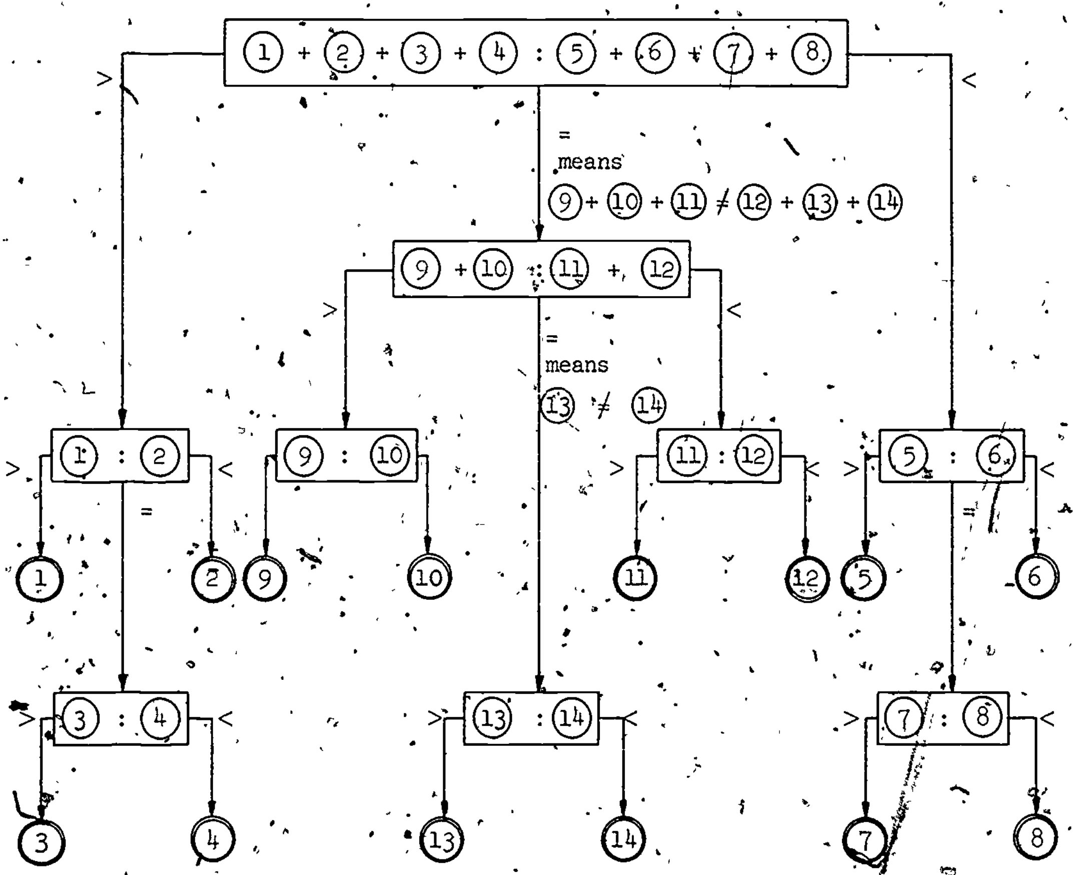
Problem 2. Weighing 12 balls with one known to be heavier.

Flow Chart Solution



Problem No. 3

Flow Chart Solution



Problem No. 4: Weighing 12 balls with one known to be either lighter or heavier.

Tabular Solution

Solution: Let the balls be A, B, C, D, E, F, G, H, I, J, K, L.
Weigh the balls as follows and note the results each time.

1st weighing	A, B, C, D	Δ	E, F, G, H
2nd weighing	A, J, G, I	Δ	C, D, E, L
3rd weighing	A, C, H, I	Δ	D, F, J, K

If the left side of a weighing was down, we write L.
If the right side of a weighing was down, we write R.
If there was a balance, we write "=".
If the results were:

L L L	A is heavy	R L =	G is heavy
R R R	A is light	L R =	G is light
L = =	B is heavy	R = L	H is heavy
R = =	B is light	L = R	H is light
L R L	C is heavy	= L L	I is heavy
R L R	C is light	= R R	I is light
L R R	D is heavy	= L R	J is heavy
R L L	D is light	= R L	J is light
R R =	E is heavy	= = R	K is heavy
L L =	E is light	= = L	K is light
R = R	F is heavy	= R =	L is heavy
L = L	F is light	= L =	L is light

Comment: It can be shown that no more than 12 balls can be searched for an "odd ball" in three weighings even though three other triples exist out of a possible 27 (= 3^3). The unused triples are = = =, L L R, and R R L. The = = = would mean that the odd ball is never on the scale and this contradicts the assumption that one of the balls is different. The triples L L R or R R L also lead to contradictions for the sequence of three weighings we have used.

Solution to the Concentration Camp Problem

Suppose that you are one of the three prisoners. You will have no objection to a cellmate receiving a piece which is no larger (in your consideration) than his share of the loaf.

If a piece of the loaf is (in your estimation) an exact share, you should be indifferent as to who gets it.

Our solution will guarantee that before you are served no one else will get a piece which is (in your estimation) greater than his share and that you will never be stuck with a piece which (in your estimation) is less than your share.

First the prisoners are numbered in order, 1, 2, and 3. Prisoner No. 1 cuts off a slice which he claims is $\frac{1}{3}$ of the loaf. He is then indifferent as to who gets it. This slice is now offered to the second prisoner. If

(a) he feels that this slice is no larger than a fair share, he rejects it;

(b) he feels that the slice is larger than a fair share, he trims it to the size of a fair share.

In either case (a) or (b) the slice is now offered to prisoner 3 who either accepts it or rejects it. If he rejects it, then it reverts to the last person who has cut or trimmed it.

Chapter 2

INPUT, OUTPUT AND ASSIGNMENT

In many respects this is the most important chapter in the book. Along with the companion chapter in the language supplement, as much as three weeks of classroom and quiz meetings may be required. The purpose of grouping the three concepts of input, output and assignment in a single chapter (which at first may seem somewhat ambitious) is two-fold.

First, input and assignment steps are closely related in that both result in the assigning of values to variables. Moreover, input is closely related to output in that one process can be thought of as the reverse of the other. Hence, all three concepts, input, output, and assignment, seem to be directly or indirectly linked.

Second, when the student completes this chapter he can draw flow charts for many simple but complete algorithms. Exercises involving the construction of some surprisingly interesting algorithms are given at the end of Sections 2-3 and 2-5. Moreover, when the student has completed the study of the companion chapter in the language manual, he can write complete programs for the algorithms. These can then be "run" and tested on the computer. It is the quickest route to laboratory practice. Here we assume that most schools will have access to a computer for running student problems.

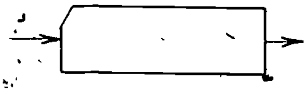
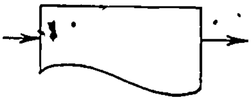
The chapter outline is as follows:

The discussions of these topics will be contained in the appropriate sections of the chapter.

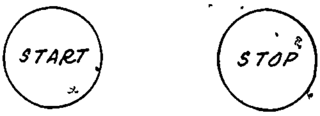
2-1 The Flow Chart Concept

We illustrate the first flow chart in the so-called flow chart language which is described in this text (Figure 2-2). A simple problem is used which embraces all three basic actions; input, assignment and output. The transition from a word problem to an algorithm in flow chart form is illustrated.

A preliminary or initial explanation is given for the three basic actions. The ideas of input and output are then repeated and elaborated. Flow chart silhouettes for each of these actions or events, i.e.,

 for input,  for output,

are also discussed in some detail. The start and stop circles

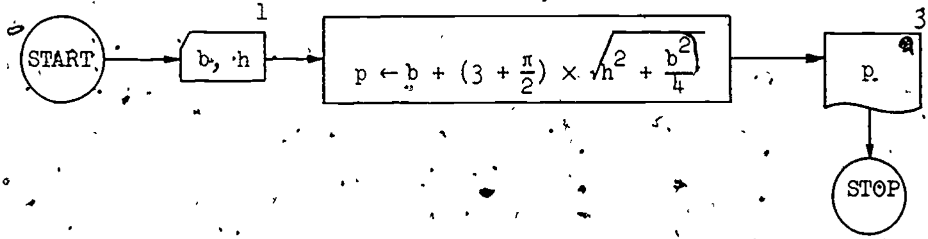


are also introduced.

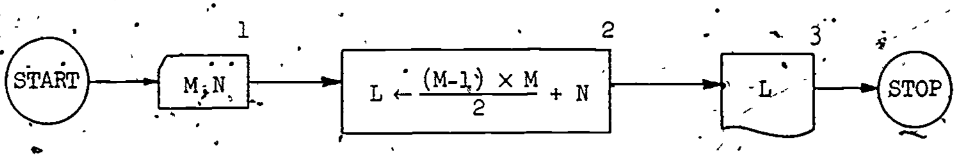
Exercises are presented in which the student converts some simple word problems into a flow chart whose structure, he is told, is similar to the problem first illustrated in Figure 2-2. This figure, incidentally, appears several times in this chapter and again in Chapter 3. A number of ideas stem from its study.

Answers to Exercises 2-1

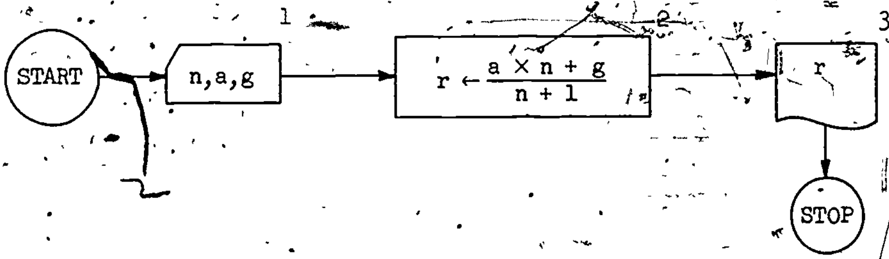
1.


START
1
b, h
2
$p \leftarrow b + (3 + \frac{\pi}{2}) \times \sqrt{h^2 + \frac{b^2}{4}}$
3
p
STOP

2.



3.



2-2 Repetition

The first illustrative problem (Figure 2-2) is now expanded to consider repetition of the simple process. A loop is formed (Figure 2-6) and explained. The student is trained to follow the loop by working his way through it several times, each time with another set of data. Opportunities are thereby afforded to cement concepts of destructive read-in and non-destructive read-out.

The distinction between an endless and a terminating loop is made. At this point we begin to repeat and expand in detail the initial idea of an assignment step. The silhouette representing assignment,

and its contents are brought into focus.

The kind of computer process it suggests is outlined in a three-step process. The section closes with a few very simple exercises to make certain the student has understood the loop just shown in Figure 2-6.

Answers to Exercises 2-2

(a)	A = 5.0	B = 10.0	C = 3.0	D = unknown
(b)	A = 8.5	B = 5.7	C = -3.2	D = 10.7
(c)	A = 5.0	B = 10.0	C = 3.0	D = 11.6
(d)	A = 4.3	B = 2.5	C = 6.1	D = 11.6

2-3 Assignment and Variables

It may well be that this is the most important section in the entire text. A much more careful scrutiny is given to the assignment step. Since assignment involves the assigning of a value to a variable, the first job is to define what we mean by a variable. In many texts on computing there is only the fuzziest relationship established between variables of the computer language described and variables as they are ordinarily defined in mathematics. In this text great pains are taken to make the two concepts essential identical or at least compatible. An analogy (Figure 2-3) is drawn between a mathematical variable and its value and a flow chart variable and its value. The latter can be thought of as a wooden box. The (mathematical) variable is engraved on its cover and its (current) value is stored inside the box. We make numerous references to this analogy in explaining other concepts at later points in the text.

With the aid of the wooden box we define assignment in detail. Later the box is visualized as having a window. To explain a read-out from memory (non-destructive) we simply send a messenger to look through the window. To read-in, or store in memory (i.e., assign a value to a variable), we send another messenger who opens the box, dumps out its contents, puts a new value inside and closes the box. This simple-minded analogy becomes the basis for good humor and, we hope, will prove to be powerful pedagogically.

Another point carefully made in this section is the distinction between the equality symbol of algebra, e.g., $x = 2$, and the assignment symbol of the flow chart language, e.g., $x \leftarrow 2$.

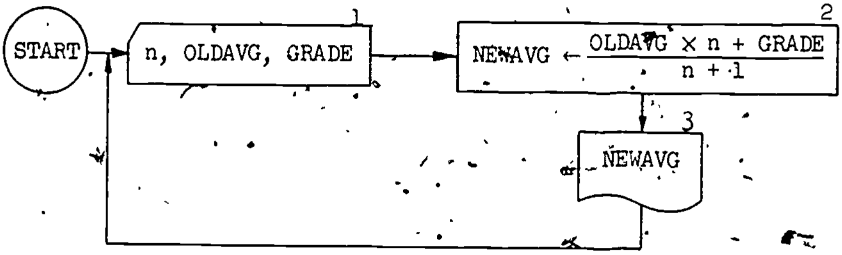
The section closes with a set of exercises for constructing simple algorithms, the first ones of some real interest. This teachers' commentary on the set is quite complete; several extensions of the exercises are discussed.

A rather simple but quite effective way to make the ideas of assigning and reading values of variables clear to all students is to construct actual working models of "window boxes" and organizing the class to carry out some algorithms using the boxes for storing values of variables. The boxes may be constructed from a shoe box, for example, with a hole cut in the side. The number in the box may be represented by writing a name of the number on a slip of paper and placing it in the box so that the name can be read through the window. When, in Section 2-6, you want to represent alphanumeric data stored in the box, you can do it by placing quotation marks around the alphanumeric data. Then it will be impossible to confuse your representation for a number with your representation for a numeral naming that number.

Answers to Exercises 2-3

Note: Problems 1, 3, 4 and 6 of this set require flow charts. Problems 2 and 5 require the student to explain something. The problem set develops sequentially. If pressed for time, a class could do only Problems 1, 2 and 3.

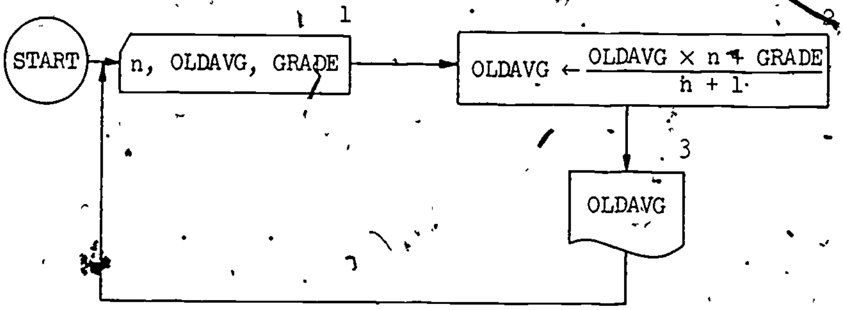
1.


START
n, OLDAVG, GRADE
1
NEWAVG ← $\frac{\text{OLDAVG} \times n + \text{GRADE}}{n + 1}$
2
3
NEWAVG

2. In the first form of the problem we used single letters for the variables. It was difficult to choose a single letter which conveyed any meaning to quantities like old average, new average, or grade. The use of character strings, even if limited to a fixed length, like a maximum of six characters, is preferable.

Without question the flow chart for the second solution would mean more than the first flow chart even if the elapsed time were only a week. After one year the first solution is likely to have little or no residual meaning while the second solution with symbols like OLDAVG, NEWAVG and GRADE is likely to retain all of its original meaning to the person who drew the flow chart (or even to someone else).

3.


START
n, OLDAVG, GRADE
1
OLDAVG ← $\frac{\text{OLDAVG} \times n + \text{GRADE}}{n + 1}$
2
3
OLDAVG

In box 2 we are "updating" the OLDAVG. Before execution, OLDAVG is the grade average based on n scores. After execution of box 2, OLDAVG is the average of n + 1 scores. The old value is not needed and can be destroyed. Storage is conserved by not using NEWAVG as a variable.

4. For Abel: n = 7, OLDAVG = 77.1, GRADE = 91.
   For Chary: n = 7, OLDAVG = 71.2, GRADE = 82.
   For Williams: n = 7, OLDAVG = 84.6, GRADE = 87.

5. Yes, because any and all individual grades can be recomputed from a knowledge of the series of cumulative averages. (A grade which is recomputed from two successive cumulative averages may be subject to some uncertainty by virtue of the roundoff errors in the average values.)

6. For any given student we can compute each successive grade as follows:

   1st grade is simply the entry in column 1, denoted by CUM 1
   2nd grade is computed by noting that
   2nd grade + 1 × entry in col. 1 = 2 × entry in col. 2
   or 2nd grade + 1 × CUM 2 = 2 × CUM 2
   or 2nd grade ← 2 × CUM 2 - 1 × CUM 1
   3rd grade + 2 × entry in col. 2 = 3 × entry in col. 3
   or 3rd grade ← 3 × CUM 3 - 2 × CUM 2

   .
   .
   .

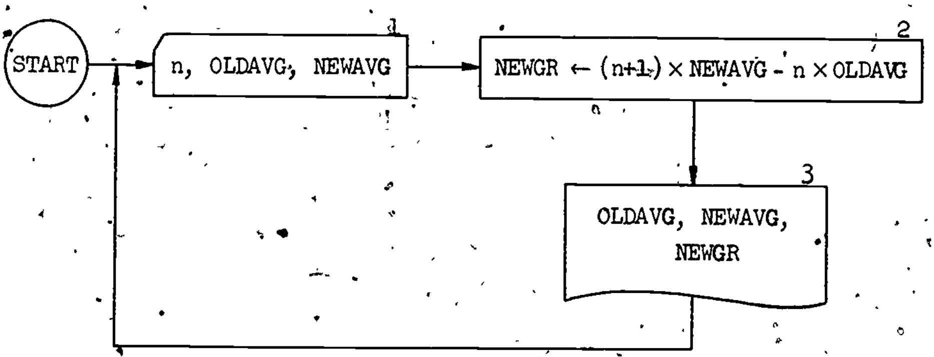
and in general

n + 1st grade ← (n + 1) × CUMNPLUS1 - n × CUMN

If we let NEWGR be the n + 1st grade
NEWAVG the average of n + 1 grades and
OLDAVG the average of n grades

then we get a flow chart like:

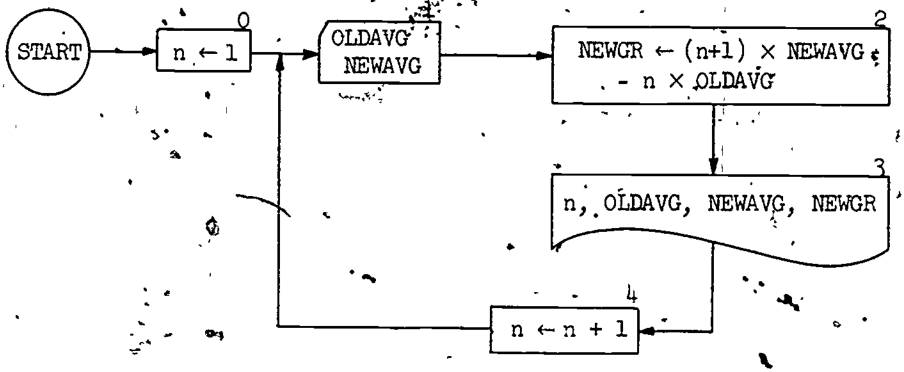
(a)



Comment

Another way to view this process is to draw a parallel with the flow chart in Figure 2-11 to obtain:

(b)

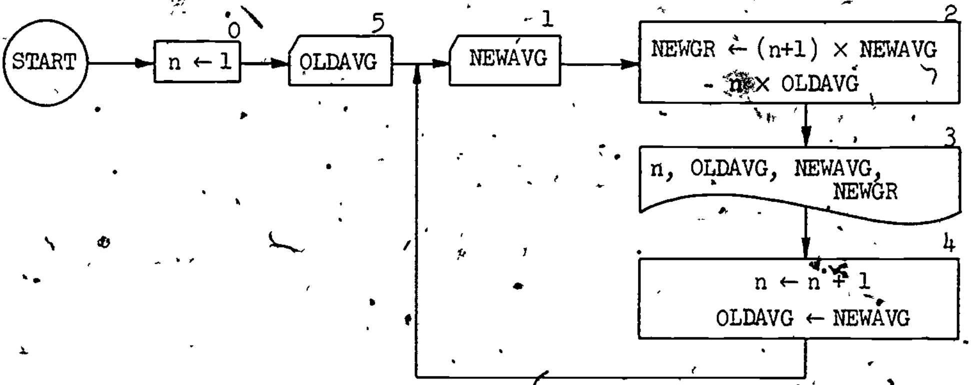


The loop in (b) more clearly suggests the computation of the series of grades beginning with the second grade.

To use flow chart (b) each data card must contain two values, the nth and the n + 1st cumulative average. Thus, for Smiley Chary, successive data cards will contain the pairs 54.0, 64.0 to compute the 2nd grade, 64.0, 67.0 to compute the third grade, etc. There is one important difference between the basic idea in Figure 2-13 and its application here in flow chart (b). Here the tally n is actually used in box 2. This was not the case in Figure 2-13.

A weakness of the approach used in (b) is the duplication of data on successive cards. Thus, the second value on one card is a duplicate of the first value on the following card. This overlap can be eliminated with the scheme shown in flow chart (c). Here only the nth cumulative average must be input to compute the nth grade.

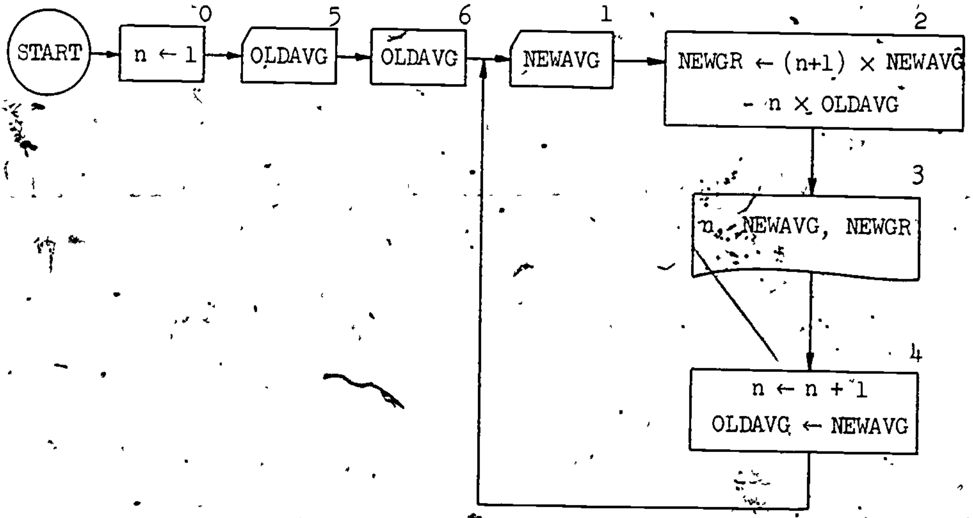
(c)



Flow chart (c) can be explained by imagining we want to recover all grades for Smiley Chary beginning with the second grade. We punch a series of data cards containing the successive grade averages for Chary, one value per card. The first card will contain 54.0 in this case. The flow chart tells us that box 5 will be executed only once. The value from the first card is assigned to OLDAVG, but the value from the next (and all succeeding cards) will be assigned to NEWAVG. Thus, after the 2nd grade is computed (box 2) and printed (box 3), box 4 provides the preparation to compute the next grade. First n is incremented. Then OLDAVG is reassigned a value equal to the current value of NEWAVG. This is really another form of "updating."

Notice that the effect of box 3 is to print a line on which the second item, OLDAVG, is a repeat of the third item, NEWAVG, on the preceding line. Do we want this duplication? Why have we taken the trouble to remove this kind of redundancy on input and not on output? We can easily delete the second item from the output list in box 3, so why don't we? One answer is that the incremental cost of printing the extra numeral is nil, while readability is somewhat enhanced because each line tells the whole story. There is one more somewhat subtle point. If we drop OLDAVG from the output list, we will have no printed record of the value on the very first data card which was read by executing box 5. This can be remedied in a number of ways--but any remedy will add one or more boxes to the flow chart--as for example, flow chart (d).

(d)



It would seem that the slower students could be challenged to develop a process like this one and even help you keep your school records this way--by computer. At a later point in the course you might discuss this data recovery or "retrieval" process from the standpoint of round-off error. It is really not possible to recover all the information originally input to the system. Grades cannot always be recomputed exactly due to round-off error in the recorded averages (the cumulative averages are only recorded to one decimal place). Nevertheless, for the instructor's pusoose, this method of grade regeneration (information retrieval) is satisfactory.

2-4 Arithmetic Expressions

Arithmetic expressions which appear on the right-hand side of the assignment arrow come in for close inspection in this section. We are especially interested in pointing out the places in "every day" mathematical notation which cause problems in interpretation due either to reading difficulty or ambiguity or both. A group of practical improvements in conventions for mathematical notation are listed as suggestions. If followed, the resulting expressions would not only be easier to read, type or print, and unambiguous mathematically, but also easy for computers to read.

Two of the suggestions are adopted for our flow chart language. These are (a) abandon the practice of using juxtaposition to denote multiplication, and (b) embrace function arguments in parentheses, like cos(x) instead of cos x.

Attention is drawn to the three kinds of minuses; binary, unary, and number-naming, which may appear in a single arithmetic expression.

Rules are given for forming arithmetic expressions starting from certain basic forms (Table 2-3).

The important question of order of computation is introduced. The rôle of parentheses and the concept of a subexpression are introduced. The concept of precedence levels for arithmetic operations is presented (Table 2-4). Next is the idea of scanning an expression from left to right in search of the next task to be accomplished in the evaluation of an expression. Finally, we arrive at a simple set of rules (Table 2-6) for explaining the step-by-step procedure to be followed in evaluating an expression (however complex).

One exception is noted and explained; the case of A^{B^C} or A↑B↑C. The section includes several exercises for practice in establishing the sequence of steps in evaluating expressions, following the procedure that has been developed.

In case your students have difficulty applying Table 2-6 to Exercises 2-4, Set A, here is one further example.

Example: The expression is:

A × (B + SIN(A × (B - C) + A × C - 3))

Tabulated values for the variables:

A	B	C
3	2	5

Display of Step by Step Evaluation

Example 2

Step No.	Action	Appearance of the Expression After each Step
	Initial appearance	A × (B + SIN(A × (B $\underset{\blacktriangle}{-}$ C) + A × C - 3))
1	Compute B - C	A × (B + SIN(A $\underset{\blacktriangle}{\times}$ (-3) + A × C - 3))
2	Compute A × (-3)	A × (B + SIN(-9 + A $\underset{\blacktriangle}{\times}$ C - 3))
3	Compute A × C	A × (B + SIN(-9 $\underset{\blacktriangle}{+}$ 15 - 3))
4	Compute -9 + 15	A × (B + SIN(6 $\underset{\blacktriangle}{-}$ 3))
5	Compute 6 - 3	A × (B + $\underset{\blacktriangle}{\text{SIN}}$(3))
6	Compute SIN 3	A × (B $\underset{\blacktriangle}{+}$ .141)
7	Compute B + .141	A $\underset{\blacktriangle}{\times}$ 2.141
8	Compute A × 2.141	6.423

We want to be sure the student is acquainted with the "facts of life" about the restrictions imposed by the way machines read expressions. They read expressions as a stream or string of characters which means that expressions must be written on a single line. We also want the student to understand rather vividly that characters as well as numbers can be stored in memory.

Regarding Figure 2-18:

The parentheses used in Case b are unnecessary, but in Case c they are necessary.

For Case b, both B/C/D and (B/C)/D would mean $\frac{\frac{B}{C}}{D}$, following the precedence and left-to-right rules we have developed for these expressions. That is, the parentheses neither change the mathematical intent nor change the computational order. Note, however, that B/(C × D), whose mathematical intent appears the same as (B/C)/D, is computationally different. Because of round-off considerations, such an alternative could yield different results.

For Case c, B/(C/D) is mathematically equivalent to B/C/D, but is computationally different. The round-off error for real numbers B, C and D may be small, but if B, C and D represent integers, the difference in computational order becomes significant.

For example, let

$$\left.\begin{aligned} B &= 4 \\ C &= 6 \\ D &= 3 \end{aligned}\right\} \text{ all integers}$$

Now (C/D) is $\frac{6}{3}$ or 2, i.e., 3 goes into 6 two times. So, B/(C/D) corresponds to 4/2 or 2. On the other hand, B/C or 4/6 is zero, i.e., 6 goes into 4 zero times. So, B/C/D corresponds to 0/3 or zero!

Answers to Exercises 2-4 Set A

Displays of step-by-step evaluations

This column not required in student solution

Exercise	Step No.	Action	Appearance of the expression after each step
1		Initially →	((a × X + b) × X + c) × X + d
	1	Compute a × X	((4 + b) × X + c) × X + d
	2	Compute 4 + b	(3 × X + c) × X + d
	3	Compute 3 × X	(6 + c) × X + d
	4	Compute 6 + c	8 × X + d
	5	Compute 8 × X	16 + d
	6	Compute 16 + d	13
2		Initially →	(a - b) × (c - d)/(e × (f + g))
	1	Compute a - b	(-1) × (c - d)/(e × (f + g))
	2	Compute c - d	(-1) × (-1) /(e × (f + g))
	3	Compute f + g	(-1) × (-1) /(e × 3)
	4	Compute e × 3	(-1) × (-1) / 9
	5	Compute (-1)×(-1)	1 / 9
	6	Compute 1 / 9	0.11111...
3		Initially →	$\frac{3.14}{2} \times r^2 - (s \times \sqrt{r^2 - s^2} + r^2 \times PHI)$
	1	Compute r^2	$\frac{3.14}{2} \times r^2 - (s \times \sqrt{100 - s^2} + r^2 \times PHI)$
	2	Compute s^2	$\frac{3.14}{2} \times r^2 - (s \times \sqrt{100 - 81} + r^2 \times PHI)$
	3	Compute 100 - 81	$\frac{3.14}{2} \times r^2 - (s \times \sqrt{19} + r^2 \times PHI)$
→	4	Compute $\sqrt{19}$	$\frac{3.14}{2} \times r^2 - (s \times 4.36 + r^2 \times PHI)$
	5	Compute r^2	$\frac{3.14}{2} \times r^2 - (s \times 4.36 + 100 \times PHI)$
	6	Compute s × 4.36	$\frac{3.14}{2} \times r^2 - (39.2 + 100 \times PHI)$
	7	Compute 100 × PHI	$\frac{3.14}{2} \times r^2 - (39.2 + 112)$
→	8	Compute 39.2 + 112	$\frac{3.14}{2} \times r^2 - 151.2$

	9	Compute r^2	$\frac{3.14}{2} \times 100 - 151.2$
	10	Compute $\frac{3.14}{2}$	$1.57 \times 100 - 151.2$
	11	Compute 1.57 × 100	$157 - 151.2$
⟶	12	Compute 157 - 151.2	5.8

Comment: Results for Exercise 3 were obtained using a slide rule. The answer using an 8 decimal digit computer with $\pi = 3.14159$ and PHI(arcsin $\frac{9}{10}$) = 1.11977, is 5.8726. Notice that step 12 is a prime source of numerical error. Here the difference is taken between two terms of like sign and comparable magnitude.

Exercise	Step No.	Action	Appearance of the expression after each step (This column not required in student solution.)
4		Initially ⟶	$\sqrt{\frac{1}{2} \times (1 + q/\sqrt{p^2 + q^2})}$
	1	Compute p^2	$\sqrt{\frac{1}{2} \times (1 + q/\sqrt{9 + q^2})}$
	2	Compute q^2	$\sqrt{\frac{1}{2} \times (1 + q/\sqrt{9 + 16})}$
	3	Compute 9 + 16	$\sqrt{\frac{1}{2} \times (1 + q/\sqrt{25})}$
	4	Compute $\sqrt{25}$	$\sqrt{\frac{1}{2} \times (1 + q/5)}$
⟶	5	Compute $q/5$	$\sqrt{\frac{1}{2} \times (1 + .8)}$
	6	Compute 1 + .8	$\sqrt{\frac{1}{2} \times 1.8}$
	7	Compute $\frac{1}{2}$	$\sqrt{.5 \times 1.8}$
	8	Compute .5 × 1.8	$\sqrt{.9}$
	9	Compute $\sqrt{.9}$	.949

5. Not counting the prior step involved in computing rsq, there are three multiplication steps saved each time this expression is evaluated.

Comment: When common subexpressions appear in a given expression, like r^2 in Exercise 3, it is frequently more efficient to compute the value of this subexpression and assign this value to a new variable, like rsq, which can then appear in the larger expression in place of the original subexpression.

Thus, if the original problem was to carry out the assignment,

$$\longrightarrow \overset{1}{\boxed{\text{AREA} \leftarrow \frac{3.14159}{2} \times r^2 - (s \times \sqrt{r^2 - s^2} + r^2 \times \text{PHI})}} \longrightarrow$$

and, moreover, if box 1 is to be repeated many times for the same value of r, a more efficient approach would be to first carry out the assignments in box 0 and then execute a revised box 1 that is inside a loop.

$$\longrightarrow \overset{0}{\boxed{\begin{array}{c}\text{POVER2} \leftarrow \frac{3.14159}{2} \\ \\ \text{rsq} \leftarrow r \times r\end{array}}} \longrightarrow$$

$$\longrightarrow \overset{1}{\boxed{\text{AREA} \leftarrow \text{POVER2} \times \text{rsq} - (s \times \sqrt{\text{rsq} - s^2} + \text{rsq} \times \text{PHI})}} \longrightarrow$$

box 1 (revised)

Students can be motivated to write efficient assignment statements after looping and iteration is introduced in Chapter 4.

Answers to Exercise 2-4 Set B

There are no superfluous parentheses in Case a. In Case b, the parentheses around (D/E) can be omitted, yielding

$$(A/B + C \times D/E)/(F \times G).$$

Notice, it is possible to make a further revision

$$(A/B + C \times D/E)/F/G$$

thereby removing a pair of parentheses, but in doing so we are changing the computational rule, though not necessarily the mathematical intent.

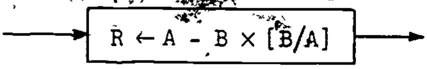
2-5 Rounding Functions

The general idea of rounding will be quite well known to the student. He knows, from long experience, that in long division and in taking square roots he must somehow terminate his process. More than likely he views rounding as a rather random procedure. The number of terms to which he carries his division, as well as the decision on the value of the last digit, is left to student caprice.

One of the purposes of this section is the presentation of rounding as a perfectly definite procedure describable in terms of well-defined mathematical functions. All of the common methods are seen, in fact, to be expressible in terms of the greatest integer function.

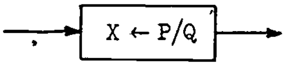
The idea that rounded values are exactly determined and that the functions used are genuine mathematical functions is reinforced when we show that these same functions are frequently used in obtaining the exact answers to certain types of mathematical problems. The only one of these which has earned a name in ordinary mathematical notation is the Greatest Integer Function, [X]. All the others are easily expressible in terms of it.

The idea of using the greatest integer function in obtaining the precise solutions to problems is illustrated in examples and problems of this section, in the Euclidean Algorithm of Section 3-2 and in numerous other examples. When such rounding is required in the mathematical solution of a problem we always indicate it in our flow charts, i.e., explicitly. Thus, in the Euclidean Algorithm of Section 3-2 we find



which assigns to R the remainder in dividing the integer A by the integer B.

On the other hand, we never indicate in our flow charts the rounding that is forced on us by the finite word length in the computer. This sort of rounding is ever-present in computing. Every arithmetic operation in a computing process is followed by a rounding action that is an application of some rounding function. If, for example, the rounding being used by the computer is ROUND to the nearest ten thousandth, then the flow chart box



will have the effect then that the value of X will be

$$10^{-4} \times \mathrm{ROUND}(10^{4} \times P/Q)$$

Again in the flow chart box

$$\longrightarrow \boxed{X1 \leftarrow (-B + \sqrt{B^2 - 4AC})/2(A)} \longrightarrow$$

one solution of a certain quadratic equation is being assigned to X1. The exact value of the expression on the right is the exact value of the desired root. However, the computer must perform nine operations in the evaluation of the right-hand expression and all but one of these operations is immediately followed by a process which is tantamount to the application of a rounding function.

It is difficult to estimate the error in the actual computed value of X1. We adopt the hopeful attitude that if the initial values computed for the variables are sufficiently accurate, then the answers we output will also be close (but less close). The dangers inherent in this assumption are strikingly brought out in Chapter 6.

It is not intended that the student should commit to memory the names of the various functions introduced in this section. The functions GRIN and FRPT should suffice for all future needs.

One more related observation;

When we want the computer to produce an exact value, we don't always get it by specifying an expression which, mathematically, is the exact equivalent of what we want.

For example: Mathematically

$$7 \times \mathrm{FRPT}(10/7) \quad \text{and} \quad 10 - 7 \times \mathrm{GRIN}(10/7)$$

are both equal to the remainder when 10 is divided by 7, but a computer with round off to 4 decimal places will compute them respectively, as

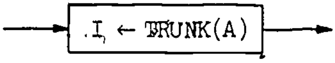
$$2.9995 \quad \text{and} \quad 3.0000.$$

The second alternative is then greatly to be preferred.

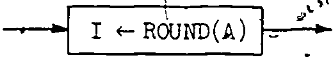
<u>Rounding to produce exact values: Flow Chart vs. Procedure Languages</u>

In languages like ALGOL and FORTRAN certain conventions are used which allow us to <u>imply</u> rather than be forced to spell out some of the rounding functions that produce exact values. We shall elaborate on this because it can be a source of confusion when carrying over key concepts like assignment from the flow chart to the procedural language.

Suppose we have a real value of A which is to be approximated as an integer in some way and suppose this approximation is to be assigned to I. We must spell out this process in the flow chart as a sequence of two steps: rounding and assignment, e.g.,



or possibly



In most procedural languages, however, the assignment statement has rounding as an <u>implied operation</u>.

Thus, in FORTRAN we would say

I = A (rounding action of TRUNK is automatic)

or in ALGOL

I:=A; (rounding action of ROUND is automatic)

Implied rounding arises because each variable has a type or mode associated with it that governs the digital coding scheme used in representing its value in storage. (Usually the type for each variable is declared as part of the program or, in the absence of explicit declaration, some "default" rule is invoked, allowing the computer to decide the type.)

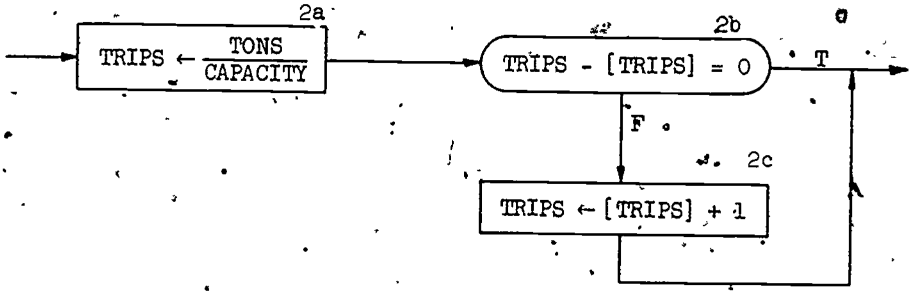
The particular rounding function that is implied in an assignment statement, as can be seen from these two examples, is unfortunately not standard. It depends on the computer language one happens to be using.

Finally, we note that <u>integer division</u>, i.e., where the remainder is thrown away, is another example of implied rounding in some of the procedural languages. Thus, in FORTRAN and MAD, there is no special operator symbol to denote the TRUNK function operating on the quotient to give its integral part. This is simply implied whenever the two operands of the division are type integer expressions.

Answers to Exercises 2-6 Set A

1. 6 trips

2. (The purpose of this question is to begin motivating an interest in Chapter 3 which deals with branching.) The algorithm is definitely not seaworthy, Wrong answers will occur when FRPT($\frac{\text{TONS}}{\text{CAPACITY}}$) = 0. I.e., when TONS is exactly divided by CAPACITY the algorithm will print out a value for TRIPS which is one too high. While this is an unlikely occurrence, it nevertheless can happen. Many students will recognize the problem but few are likely to see how to solve it, because we have not yet introduced the condition box in our flow chart language. A simple solution, which will become meaningful after completing Section 3-1, is as follows:



A bit of philosophy: Complex computer programs often have cases like this which fail to show erroneous results during the check-out phase with the particular set or sets of test data. Much later, when the program is assumed to be thoroughly tested and in actual "production", an obvious blunder in the program, like the one illustrated here, comes to light when using perfectly legitimate data.

Answers to Exercises 2-4 Set B

This group of problems gives some practice with the four rounding functions, ROUND, ROUNDUP, TRUNK, and [ ]. Exercises 8 and 9 introduce the use of the greatest integer function in modular arithmetic (computation of residues). Exercise 9 is the first of several appearances of the carnival wheel problem. Over a series of exercises in Chapters 2, 3, and 4, we show the student how this simple problem can grow in complexity from computing the winnings on a single spin of a wheel that has a simple (linear) win-loss function, to an experiment in which a wheel whose win-loss function can be of arbitrary complexity is put through a sequence of spins with the wins (or losses) being accumulated.

It is possible to develop this even further. Additional discussion is given in the commentary in connection with Exercise 4 of Section 4-1.

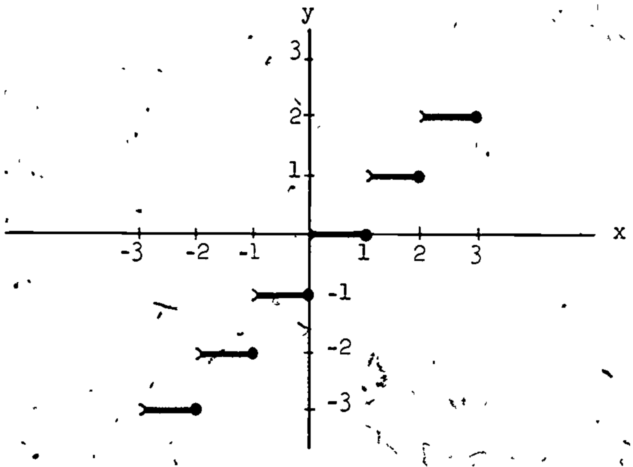
Answers to Exercises 2-6 B

1. COST = .08 × ROUNDUP(WT) or COST = .08 × (- [-WT])

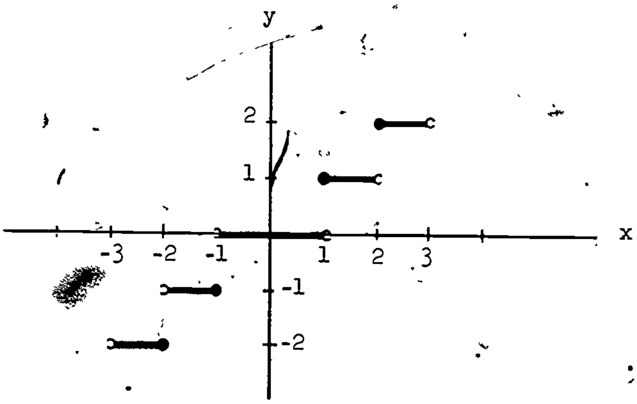
2. $N = \text{TRUNK}(\frac{\text{NBOY}}{9})$ or $N = \text{GRIN}(\frac{\text{NBOY}}{9})$ or $N = [\frac{\text{NBOY}}{9}]$

3. y = ROUNDUP(X)

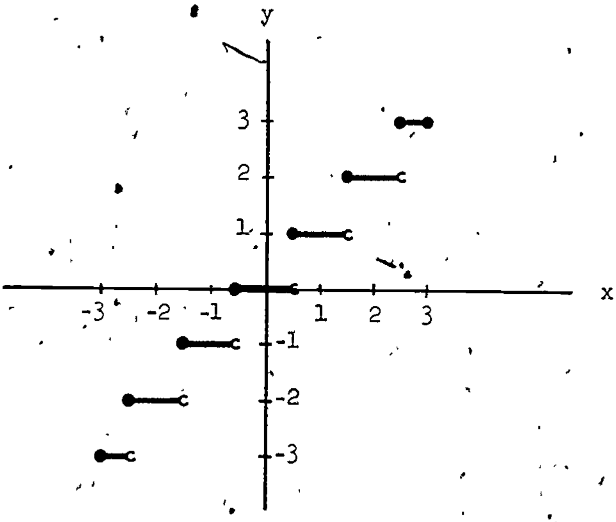
y = -[-x]



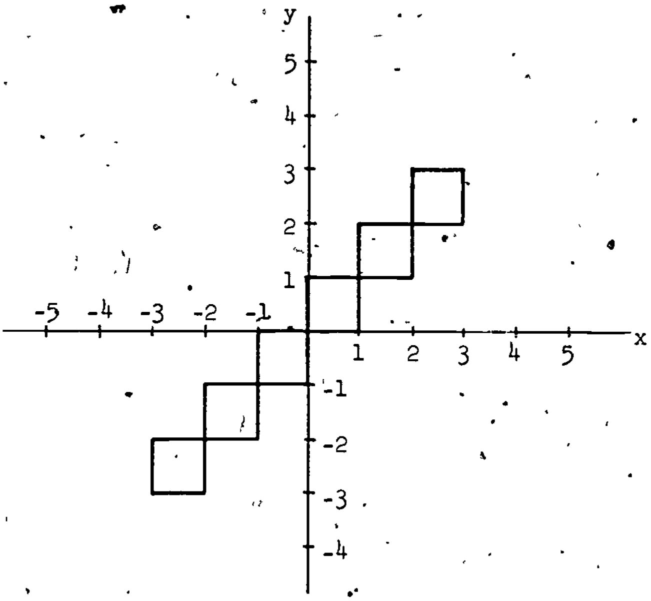
4. y = TRUNK(X)



5. y = ROUND(X).



6. Comment: This exercise is in the "fun and games" category.



7.

X	...	$-\frac{5}{2}$	$-\frac{3}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	$\frac{3}{2}$	$\frac{5}{2}$	...
ROUND(X)	...	-2	-1	0	1	2	3	...

8. a. NEWS ← S + M - $[\frac{S+M-1}{5}] \times 5$

b. None

c. NEWS ← S + M - $[\frac{S+M-1}{k}] \times k$

9. This exercise with the carnival wheel should provide good practice with modular arithmetic and should develop an appreciation of the value of integer arithmetic in general. Other exercises are built on this one in later chapters.

Answer: Notice that the wheel will come to rest at a position

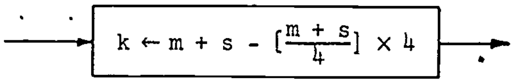
$$s_{new} = (m + s_{old}) \text{ modulo } 32$$

Then $k = s_{new}$ modulo 4.

On the other hand, we are not really interested in knowing s_{new} in this case. We only need k, which we can compute directly:

$$k = (m + s_{old})\text{modulo } 4 .$$

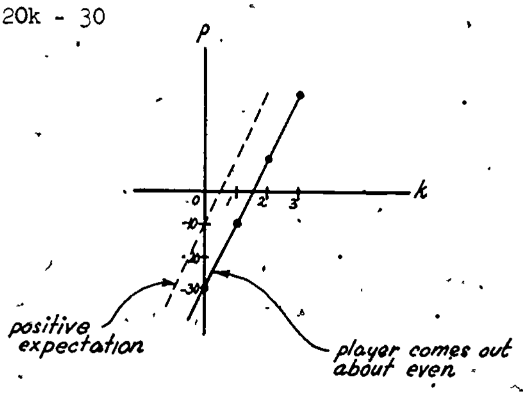
This can be expressed as the flow chart box



The points p can then be expressed in terms of k as

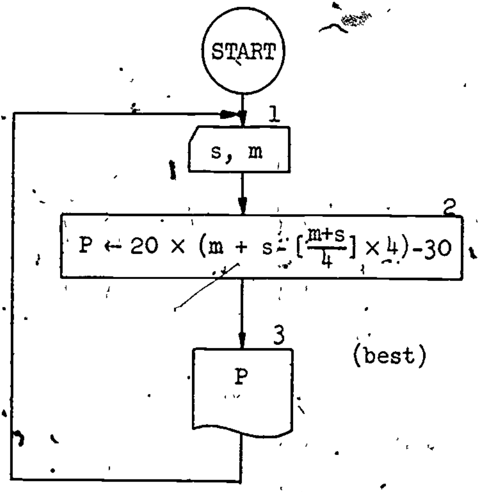
$$p = 20k - 30$$

as suggested by the graph at right which shows the four possible point values falling on a straight line.



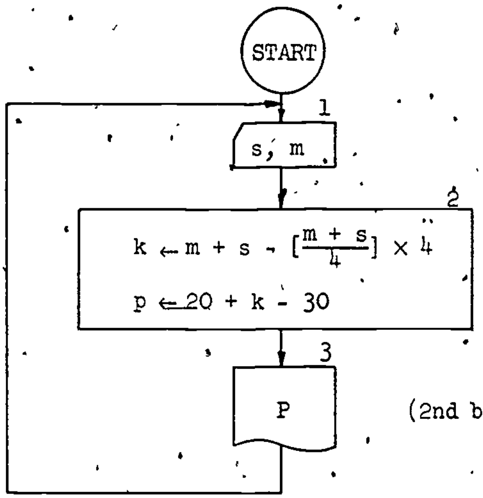
Based on the above discussion, we then have these three alternative flow chart solutions, in decreasing order of elegance.

(a)


START
1
s, m
2
$P \leftarrow 20 \times (m + s - [\frac{m+s}{4}] \times 4) - 30$
3
P

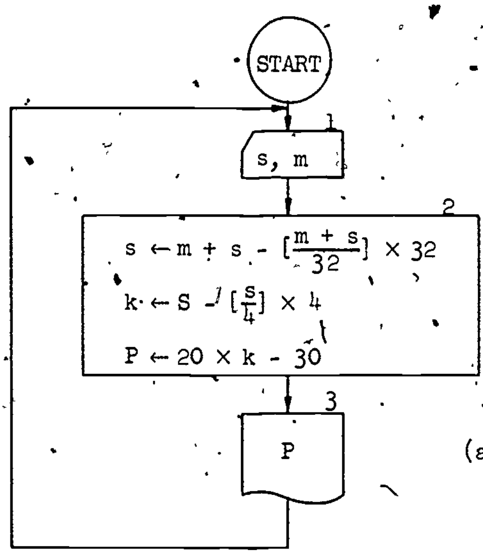
(best)

(b)



(2nd best)

(c)



(acceptable)

If you feel some or all of your students will need additional help with this problem, you might offer some help along the following lines.

<u>Additional</u> <u>suggestions</u>

1. Suppose you divide each of the blue sector numbers by 4. What are the remainders? What are the remainders if you had started with the green sector numbers, or the red, or the yellow sector numbers? Are the remainders you just found the same as the values of k ?

2. Prepare a chart for plotting the points won versus the remainders of step 1. Call the remainders k.

3. Plot the four given point values on this chart as a function of k.

4. Do the function values fall on a straight line? If so, fihd an equation of the form

$$p = m \times k + b$$

by determining m and b.

Comment: In the problem statement the point formula is such that after a large number of spins the player should come out about even. (Coming out even means with 0 total points.) Since most players enjoy winning, it is of interest to note that shifting the straight line in the graph upward, as suggested in the figure, will give the player a positive expectation. It would be interesting to see how the students would answer the following additional question:

"How would you alter your flow chart if the rule is to be

player loses 40 points for blue
player loses 20 points for green
player wins 0 points for red
player wins 20 points for yellow?"

The answer:

Only one box needs to be changed in the flow chart--box 4 becomes

$$\longrightarrow \boxed{p \leftarrow 20 \times k - \overset{2}{\underline{\underline{40}}}} \longrightarrow$$

↑
This number is the only change!

Later, when conditional statements and subscripted variables are introduced it will be possible to simulate much more interesting games of this type.

2-6 Alphanumeric Data

The ideas developed in the preceding section, coupled with those in Section 1-4 on bit patterns for alphanumeric characters, are reviewed in order to introduce the idea that flow charts may describe the input, assignment, or output of alphanumerical as well as numerical data. Careful parallels are established between each of the three step types carried out on numerical and non-numerical data.

First we explain how a variable may have a non-numerical value. Thus, the value of X might be the letter "X". Then we show the two possible forms of an assignment step:

variable ← variable

or

variable ← alphanumerical constant

Several example flow charts which deal with alphanumerical data are illustrated and explained. The section closes by pointing out certain possibilities for ambiguity in a flow chart where it is not possible, looking only at the flow chart, to tell whether a given variable is to have numerical or non-numerical values. The ambiguity is eliminated in one way or another in each computer programming language.

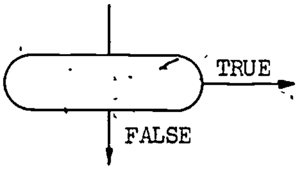
Comment: We believe it is very important that the student grasp the potential of computers, flow charts, and programming languages for solving problems which deal with alphanumerical data, e.g., data processing, symbol manipulation, etc. But we also realize that there simply may not be enough time in a one-semester course to cover this topic and those that bear somewhat more directly on conventional mathematics. For this reason we have attempted to place the topics on alphanumeric data processing at the very end of this chapter and at the end of Chapter 5 where it is discussed. In this way the material may be skipped without loss of continuity. The exception, of course, is Chapter 8 which deals entirely with problems using alphanumerical data.

There are no exercises for this section. As noted in the outline we gave for Chapter 2, this section, and its companion section in the language supplement, can be skipped without loss of continuity. The section is not at all difficult to understand--it has eye-opening ideas. We hope it will prove both amusing and intriguing, and that time will be found for it.

Chapter T3

BRANCHING AND SUBSCRIPTED VARIABLES

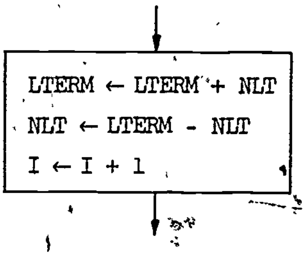
3-1 Branching is introduced. The use of the simple condition box



is illustrated with several examples. Calculation of D^2 and the comparison $D^2 \leq 841$ would suffice in Figures 3-1 and 3-2 saving computing time in taking square roots. However, the intention here is to focus on the condition box rather than efficiency. Simple programs and loops are described. Relational expressions introduced as the basis for a branch; other decision criteria are illustrated. The problem of finding and identifying the largest of three numbers is discussed in detail, showing alternative flow chart forms. The section closes with an example algorithm for tallying test scores which (Figure 3-7) uses all concepts developed to this point.

Two exercise sets are given. Set A is quite easy. It is intended to show examples of simple branching. Exercises in Set B will be more interesting to the student since he is being asked to synthesize flow charts for simple algorithms.

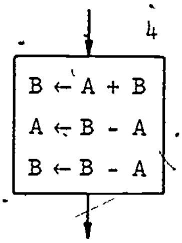
3-2 Auxiliary Variables; that is, variables which do not specifically occur in the problem statement are discussed and illustrated with significant mathematical examples. The first of these is the Fibonacci Sequence. The need for an auxiliary variable called COPY is illustrated graphically. Actually, this need for COPY can be circumvented by replacing box 4 of Figure 3-13 by



It is hard to find really simple examples in which the use of an auxiliary variable (for all its utility) cannot be avoided. Exercises using subscripted variables (Section 3-5) will require frequent use of auxiliary variables.

Exercise Set A follows the treatment of the Fibonacci Sequence. Exercise 2 uses the simple generation process of the Fibonacci Sequence to produce a set of numbers uniformly distributed between 0 and 999 (see Answers to Exercises). This exercise is referred to in later sections as a source of "random numbers".

In the same section we develop two algorithms for computing the greatest common divisor (g.c.d.) of two numbers (Figures 3-14 and 3-15). A number of later exercises will require the use of a g.c.d. algorithm. The notion of interchange of two variables, a common process, again illustrates the use of an auxiliary variable. Curiously, the need for an auxiliary variable can again be circumvented (but at some expense in complexity and computing speed). For example, instead of box 4 in Figure 3-14, we could use

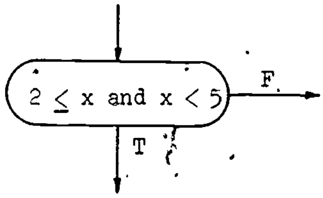


We do not feel that this is important but don't be surprised when some students come up with it.

The section closes with an introduction to the idea of an organized trace through the steps of an algorithm for purposes of "desk-checking" or verifying an algorithm. Two exercise sets are given; Set B, containing two questions, and Set C, each exercise of which requires construction of a flow chart modeling some analysis of a straight line (or line segment) through (or between) two given points.

3-3 Compound Conditions and Multiple Branching

Although we do not introduce the logical operators and, or, and not in any formal way in this text, we use these informally. For example, the compound condition



is explained, and we show the student how such "compound conditions" may be decomposed into a group of one or more simple condition boxes, each involving only one relational expression. We also show how a multi-way branch can be decomposed, if desired, into a series of simple (or two-way) condition boxes, and we illustrate this multi-way decision box in a revised form of the tallying algorithm (Figure 3-20).

There is an important set of exercises at the end of this section. Compound conditions are given in the form of verbal statements and graphs of geometric regions. The student must then develop or synthesize detailed flow charts using simple boxes for each case. He is then given several flow charts of compound conditions and asked to graph the region determined by these conditions. The carnival wheel problem from Chapter 2-6 is expanded in this set (Exercise 11) and is treated one more time in an exercise in Section 3-5, Set A.

3-4 Precedence Levels for Relations

We explain how relational symbols may be thought of as having a precedence level with respect to those of the arithmetic operators (Table 3-2). We actually avoid calling these relational symbols "operators," although, of course, most computer and language people prefer to think of them this way. We have also avoided referring to a relational expression as a "logical" or Boolean expression (whose value is either true or false). We have chosen to avoid the complication, in this text, of defining Boolean variables and Boolean values. Instead, we simply think of a relational expression as having the three-part form:

arithmetic expression	relational symbol	arithmetic expression

No exercises are given at the end of this section.

3-5 Subscripted Variables, the second crucial concept of this chapter, is introduced here. A good deal of attention is focused on the distinction between ordinary variables and subscripted variables and the distinction between inscriptions like

$$X_3 \quad \text{and} \quad X_N$$

The problem of finding the largest value in a list of values is illustrated with and without the use of subscripted variables (Figure 3-22).

A special notation is given for the input and output of data values for a list of sequentially subscripted variables. This is a notation which is in common use. Some programming languages use essentially this notation while others (notably FORTRAN) unfortunately interchange the order of the increment and the cut-off point. We can only remark that the order used in FORTRAN is not the common mathematical usage.

The section closes with the development of a very simple algorithm for sorting a group of numbers in ascending order (Figure 3-27). This algorithm illustrates nearly all the key ideas developed in the text to this point. Although Chapter 4 again takes up the question of sorting, an exercise at the end of this section (Set B) requires the student to begin thinking seriously about the process (Figure 3-27) and understand it thoroughly.

A vector is defined in this section as a list of variables like X_1, X_2, etc. of a linear array. We also agree to call the list of values of these variables a vector.

3-6 Double Subscripts are introduced here. The singly subscripted variable concept is expanded using the now familiar window box technique (see Figure 3-30). A set notation is introduced for input-output of a complete matrix.

A set of matrix elements are listed in a sequence determined by an inner index which ranges over a set of values for each value in the range of an outer index. Thus, in the notation:

$$\{\{B_{i,j}, \underset{\substack{\uparrow\\ \text{inner}\\ \text{index}\\ \text{(rows)}}}{i} = 1(1)3\}, \underset{\substack{\uparrow\\ \text{outer}\\ \text{index}\\ \text{(columns)}}}{j} = 1(2)10\}$$

the sequence of elements suggested by this list is the elements of the odd-numbered columns (one column after the other) of the B array, i.e.,

$$B_{1,1}, B_{2,1}, B_{3,1}, B_{1,3}, B_{2,3}, B_{3,3}, B_{1,5}, B_{2,5},$$
$$\ldots, B_{1,9}, B_{2,9}, B_{3,9}$$

Similarly, in the notation

$$\{\{T_{i,j}, \underset{\substack{\uparrow\\ \text{inner}}}{j} = 1(2)5\}, \underset{\substack{\uparrow\\ \text{outer}}}{i} = 1(1)3\}$$

we refer to a row-by-row sequence (3 rows) in each of which we take the elements from odd columns, i.e.,

$$T_{1,1}, T_{1,3}, T_{1,5}, T_{2,1}, T_{2,3}, T_{2,5}, T_{3,1}$$
$$T_{3,3}, T_{3,5}.$$

A simple zero-sum game using a 6 × 6 array of simple integers is then devised and modeled or translated into flow-chart form. The power of subscripts is suggested by our ability to generalize this game to one which uses an n × n array.

Several exercises requiring the student to synthesize loops which carry out simple operations on a row or column of a matrix are then given.

Answers to Exercises 3-1 Set A

1. $2 \times 5 < 7$ is false. Output value for LRGR is 5.

2. $2 \times (-3) < -5$ is true. Output value for LRGR is -5.

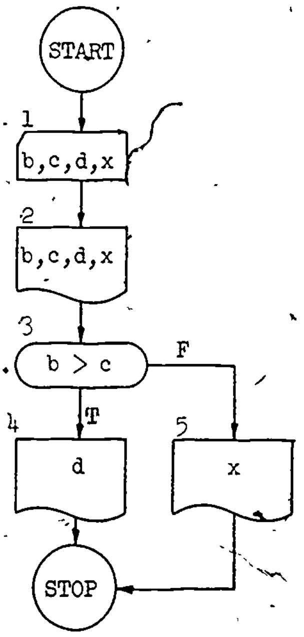
3. $2 \times (10) < 5$ is false. Output value for LRGR is 10.

	A	B	A^2	$A^2 - B$		B^2		Value printed for LRGR
4.	5	7	25	18	$\leq$	49	true	7
5.	-3	-5	9	14	$\leq$	25	true	-5
6.	10	5	100	95	$\leq$	25	false	10

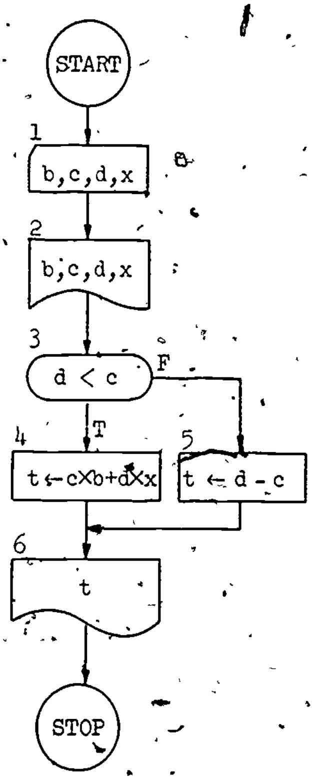
Comment on Exercises 7-11

For each exercise test data and answers are given. If you assign the computer-checking of these exercises, you may wish to give the students these data and check their computed results against the given answers.

7.



8.



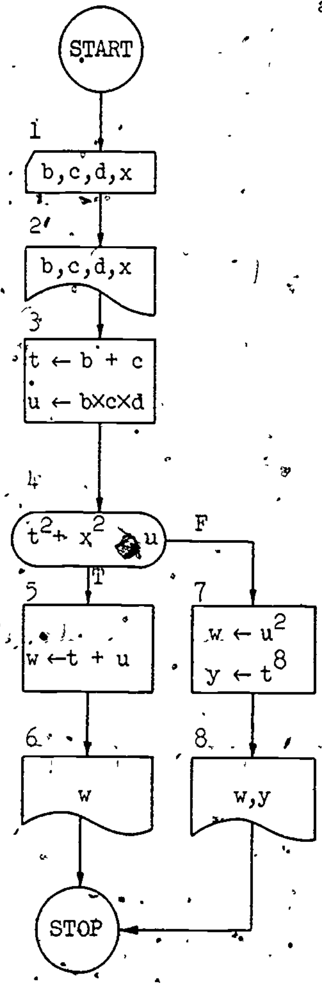
Test Data	Input b,c,d,x	Box 4 or 5 Output
1	7,3,4,2	4
2	3,7,4,2	2

Test Data	Input b,c,d,x	Box 6 Output t
1	7,3,4,2	1
2	7,4,3,2	34

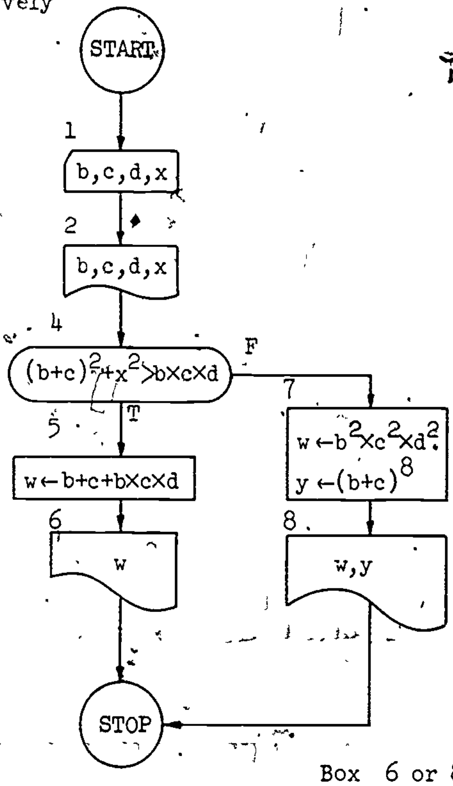
Comment on Exercise

The student may not yet see that values for $c \cdot b + d \cdot x$ and $d - c$ can each be assigned to the same variable, say t. This should be pointed out to him. The advantage of using the same variable is that a common print statement box 6 can then be used.

9. (a)



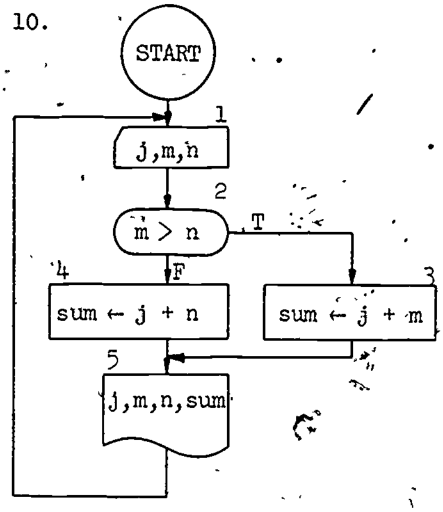
alternatively (b)


START
1
b,c,d,x
2
b,c,d,x
4
$(b+c)^2+x^2>b\times c\times d$
F
T
5
w←b+c+b×c×d
7
$w \leftarrow b^2\times c^2\times d^2$
$y \leftarrow (b+c)^8$
6
w
8
w,y
STOP

Test Data	Input b,c,d,x	Box 6 or 8 Output w	or w, y
1	7,3,4,2	94	___
2	7,3,6,2	___	$15876, 10^8$

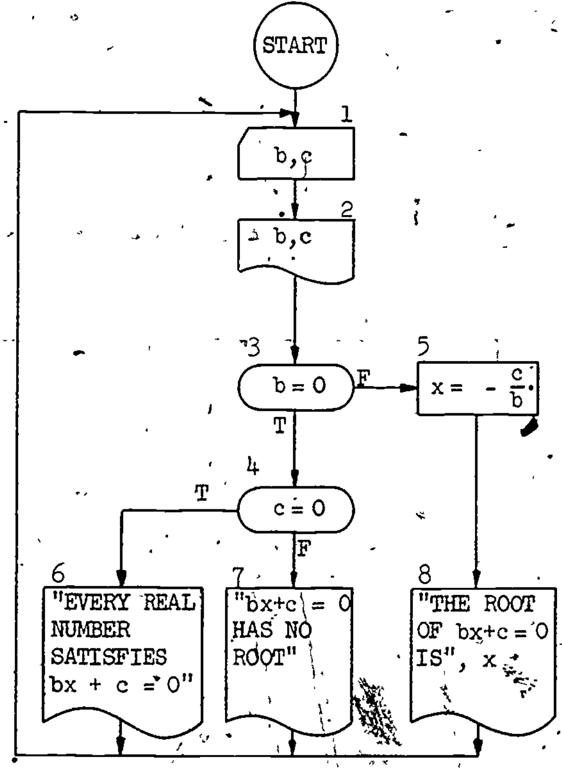
Comment, Solution (a) is more efficient computationally in the sense that subexpressions developed and temporarily stored for use in the test in box 4 are reused in boxes 5 and 7. A minimum of computation in boxes 5 and 7 is then required. The virtue of solution (b) is that it follows the statement of the problem more closely. You can expect [redacted] to turn in solutions like (b), but solutions like (a) should be pointed out to them.

10.



Test Data	Input j,m,n	Box 5 Output j,m,n,sum
1	7,9,11	7,9,11,18
2	8,11,9	8,11,9,19

11.

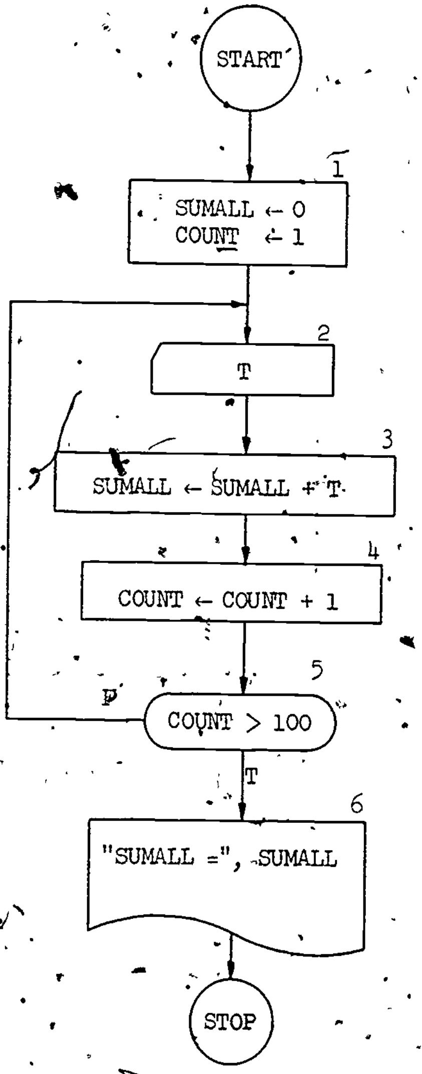


Test Data	Input b,c	Box 6, 7, or 8 Output
1	2,4	the root of bx + c = 0 is -2.0
2	0,4	bx + c = 0 has no root
3	0,0	every real number satisfies bx + c = 0

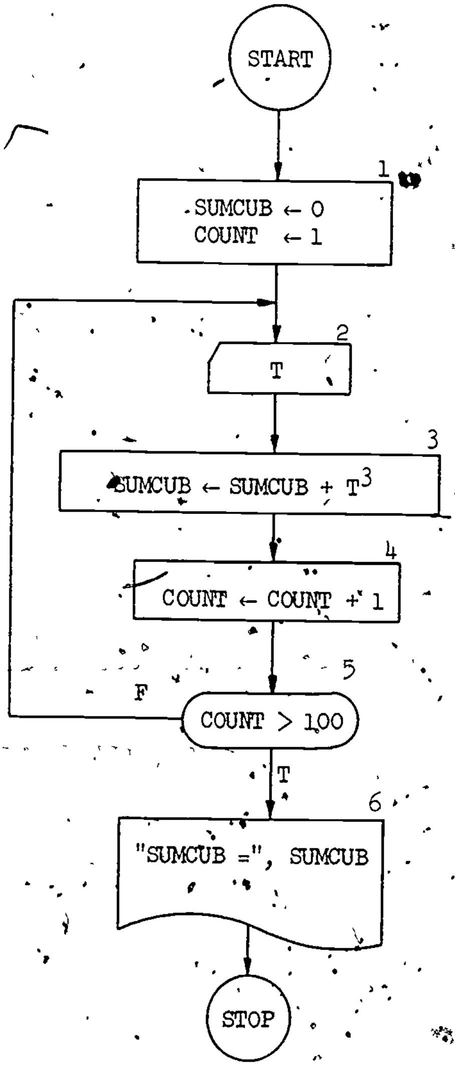
Exercises 3-1 Set B

This is a very important exercise set. We expect the student to try his hand at synthesizing simple loops. Students should be asked to do at least two or three exercises here including 4 and 5. If necessary, the answer to 1 could be given to them. But, if so, he should then be asked to do all the rest.

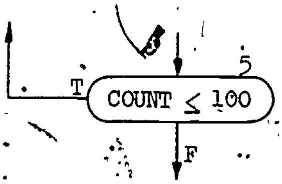
1.



2.

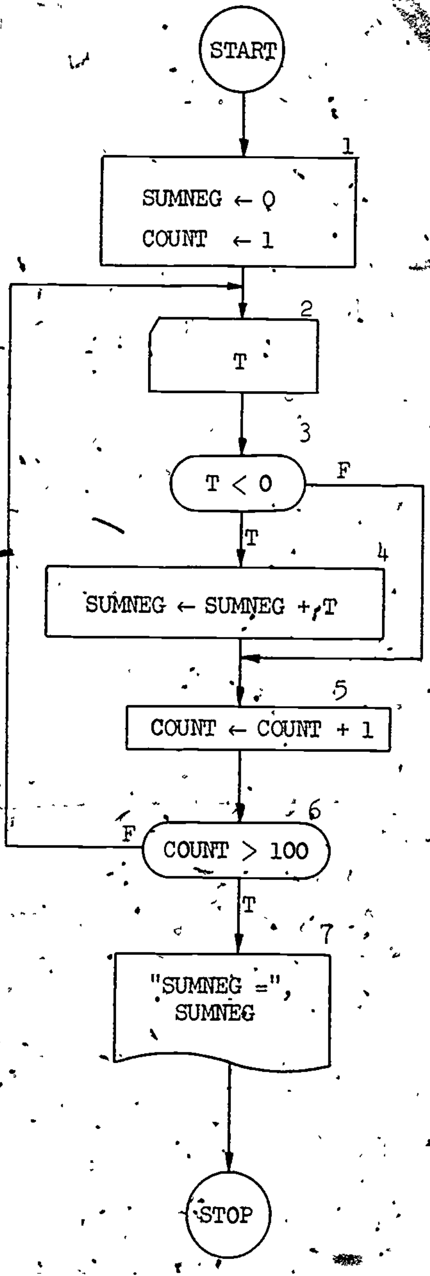


Alternate solution:
Box 5 could be

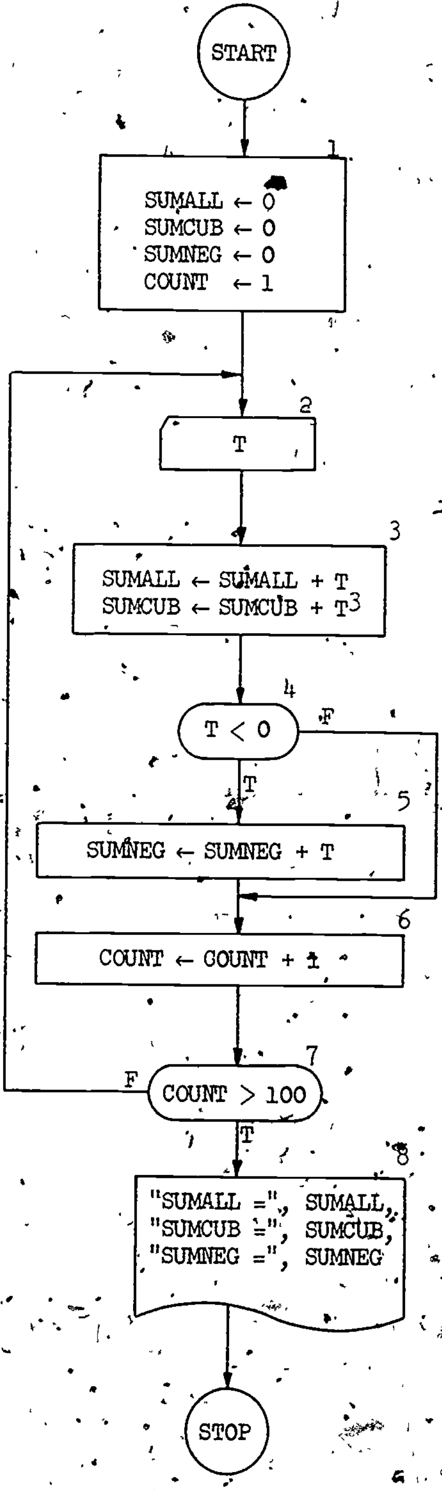


T3

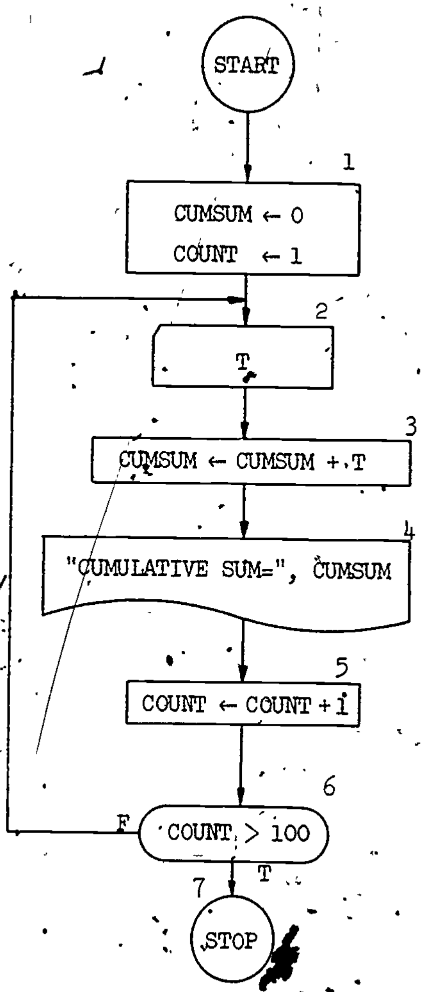
3.


START
1
SUMNEG ← 0
COUNT ← 1
2
T
3
T < 0
F
T
4
SUMNEG ← SUMNEG + T
5
COUNT ← COUNT + 1
6
F
COUNT > 100
T
7
"SUMNEG =",
SUMNEG
STOP

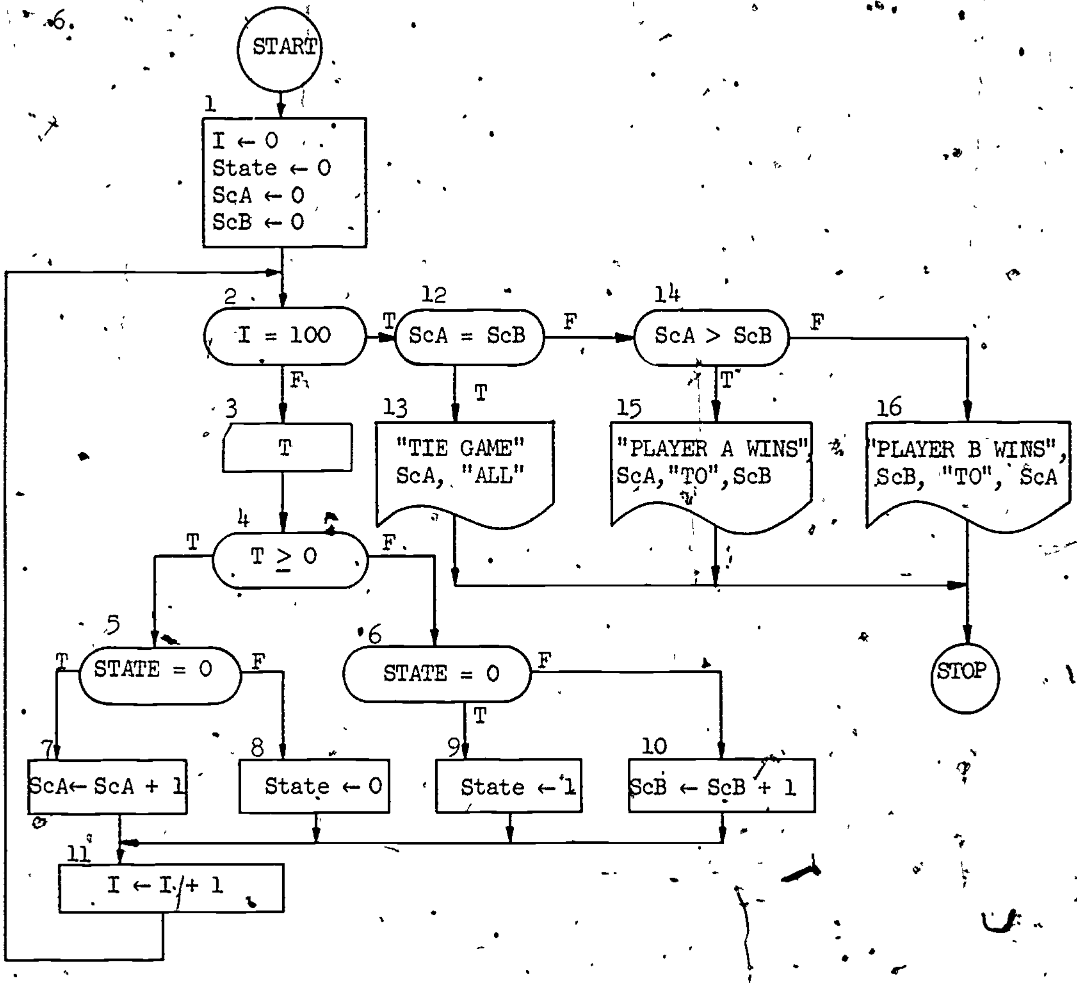
4.


START
1
SUMALL ← 0
SUMCUB ← 0
SUMNEG ← 0
COUNT ← 1
2
T
3
SUMALL ← SUMALL + T
SUMCUB ← SUMCUB + T^3
4
T < 0
F
T
5
SUMNEG ← SUMNEG + T
6
COUNT ← COUNT + 1
7
F
COUNT > 100
T
8
"SUMALL =", SUMALL,
"SUMCUB =", SUMCUB,
"SUMNEG =", SUMNEG
STOP

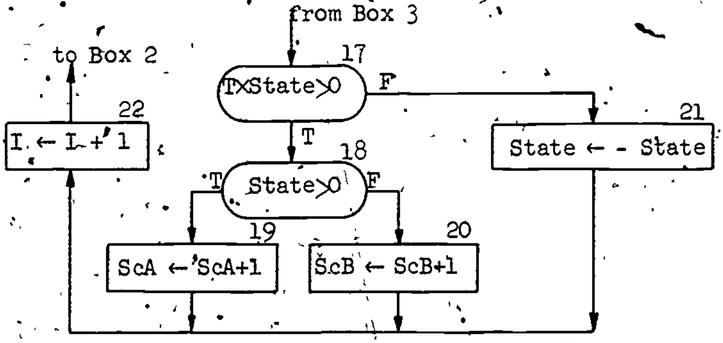
5.


START
1
CUMSUM ← 0
COUNT ← 1
2
T
3
CUMSUM ← CUMSUM + T
4
"CUMULATIVE SUM=", CUMSUM
5
COUNT ← COUNT + 1
6
F
COUNT > 100
T
7
STOP

6.



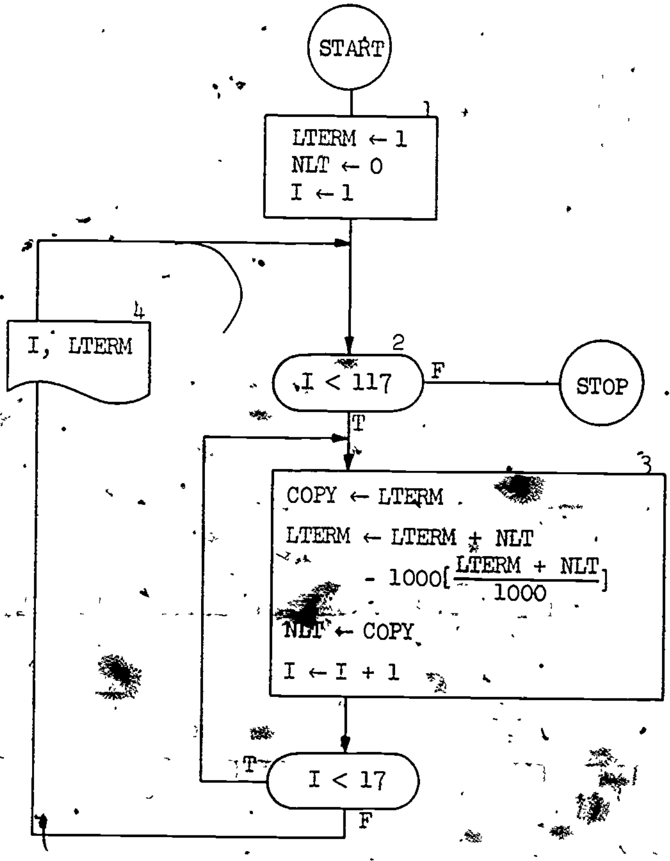
Comment: The condition boxes in the solution could, of course, be arranged in a variety of equivalent ways. A more subtle solution, not likely to be produced by the student at this stage, is shown below. We use 1 and -1 for the states in order to simplify box 17.



Answers to Exercises 3-2 Set A

1. We assign 0 for the initial value of NLT instead of 1 for the following reason: The values printed for LTERM at box 2 would be 1, 1, 2, 3, etc. But, if NLT were initially assigned the value 1, then values printed for LTERM would be 1, 2, 3, 5, etc., with the first number of the series missing.

2.

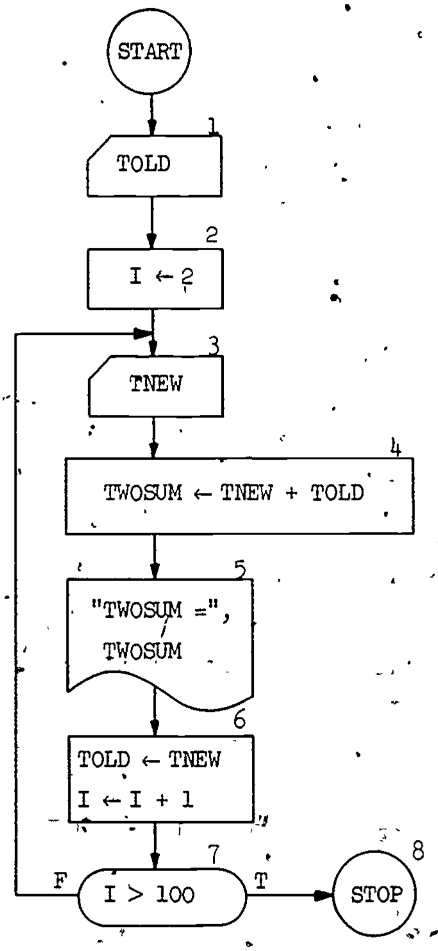


For large numbers in the Fibonacci sequence the n rightmost decimal digits are essentially uniformly distributed between 0 and 10^n. Thus, an algorithm like the one for this problem has been found useful for generating random numbers, especially when n is larger. (See, for example, B.A. Galler, "The Language of Co[illegible]ers," Section 6.3, p. 72.) Even for n equal three it may be poss[illegible]le to see that the numbers approach a uniform distribution.

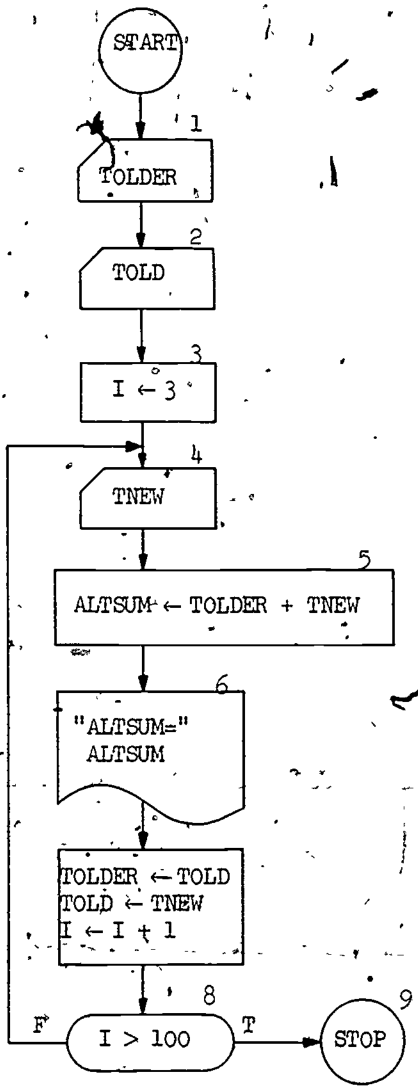
I.e., approximately half of the generated numbers should exceed 500 and approximately one-tenth of the 100 generated numbers should fall between 300 and 400.

Suggestion: You might use the 100 generated numbers as the data set for the exercises in Section 3-1, Set B.

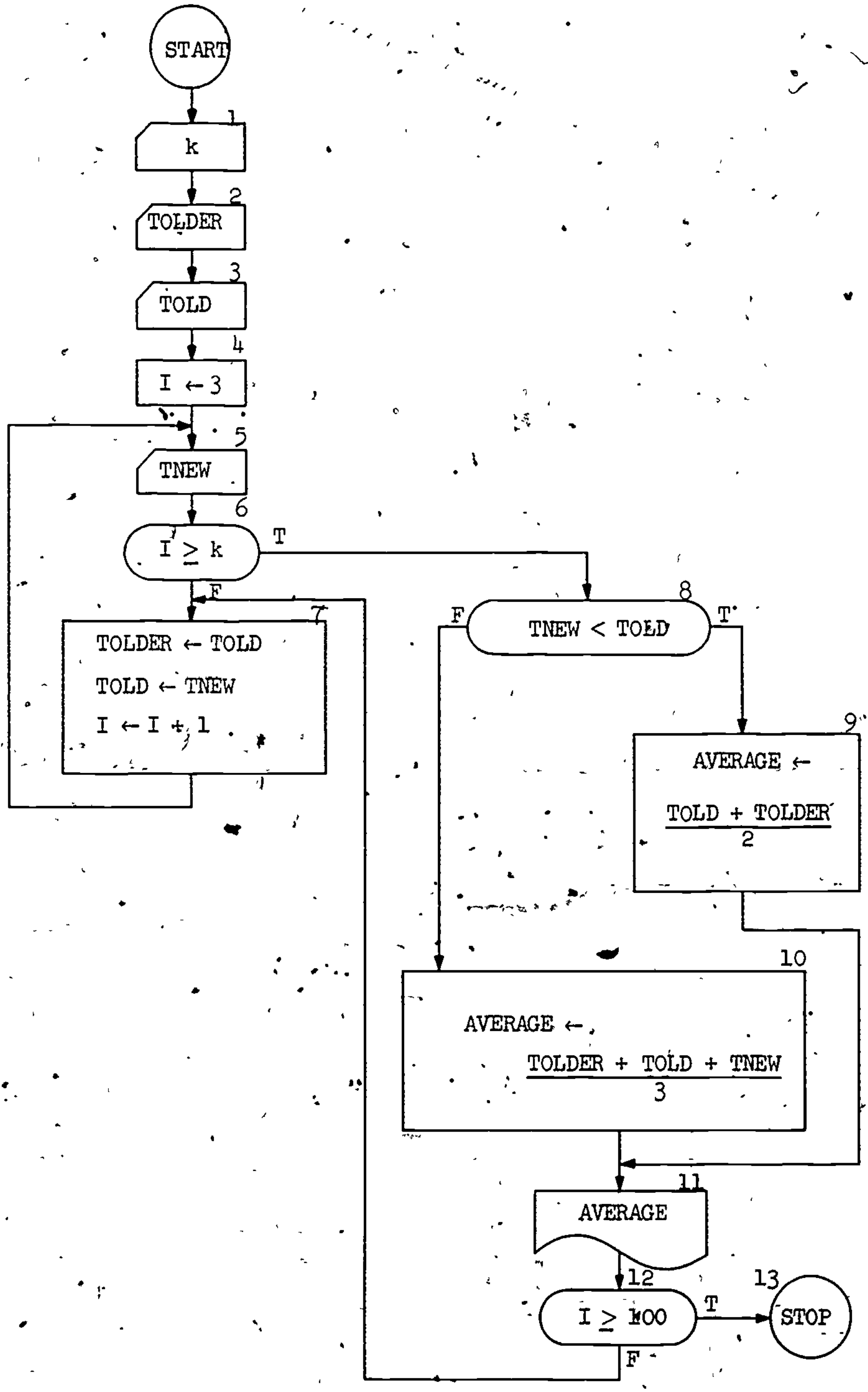
3.


START
1
TOLD
2
I ← 2
3
TNEW
4
TWOSUM ← TNEW + TOLD
5
"TWOSUM =", TWOSUM
6
TOLD ← TNEW
I ← I + 1
7
I > 100
F
T
8
STOP

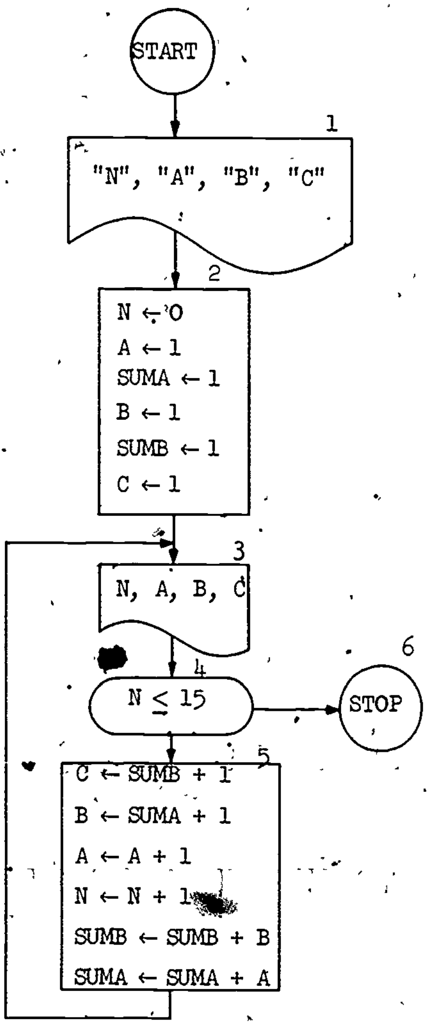
4.


START
1
TOLDER
2
TOLD
3
I ← 3
4
TNEW
5
ALTSUM ← TOLDER + TNEW
6
"ALTSUM="
ALTSUM
TOLDER ← TOLD
TOLD ← TNEW
I ← I + 1
8
I > 100
F
T
9
STOP

5.


START
1
k
2
TOLDER
3
TOLD
4
I ← 3
5
TNEW
6
I ≥ k
T
F
7
TOLDER ← TOLD
TOLD ← TNEW
I ← I + 1
8
TNEW < TOLD
F
T
9
AVERAGE ←
TOLD + TOLDER
2
10
AVERAGE ←
TOLDER + TOLD + TNEW
3
11
AVERAGE
12
I ≥ 100
T
F
13
STOP

6.



Comments:

You may need to give the students some hints in order to prevent them from summing all the terms in Column A and Column B each time they need a new B and C, respectively. The hint might go something like:

> "Summing all the terms in Columns A and B each time you need to compute a new B and C, respectively, is wasteful, difficult to program and can be easily avoided if you employ auxiliary variables whose values are the "running" sums of Columns A and B."

Exercise 6 calls for the generation of the following table:

N	A	B	C
0	1	1	1
1	2	2	2
2	3	4	4
3	4	7	8
4	5	11	15
5	6	16	26
6	7	22	42

etc.

by the process

$$B_N = 1 + \sum_{i=0}^{N-1} A_i$$

$$C_N = 1 + \sum_{i=0}^{N-1} B_i$$

Notice also that

$$B_N = A_{N-1} + B_{N-1}$$

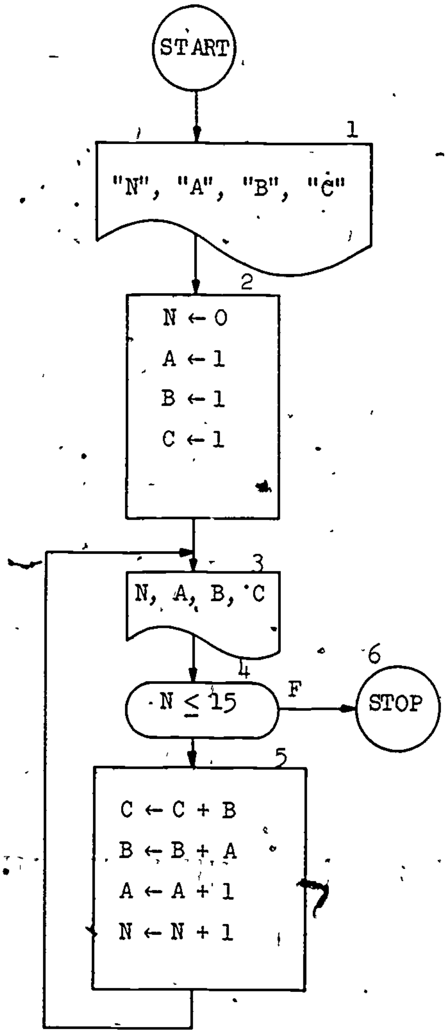
$$C_N = B_{N-1} + C_{N-1}$$

for all the entries shown. Is it generally true? Consider

$$\begin{aligned} B_{N+1} &= 1 + \sum_{i=0}^{N} A_i \\ &= 1 + A_N + \sum_{i=0}^{N-1} A_i \\ &= A_N + \left(1 + \sum_{i=0}^{N-1} A_i\right) = A_N + B_N \end{aligned}$$

which proves the general truth of the second form (for the B) by mathematical induction.

The solution to the same problem is now seen to be:



Additional remarks on the Fibonacci sequence

In this Chapter and in Chapter 4 we have made considerable use of the Fibonacci Sequence. This sequence has served as a pedagogical vehicle for introducing and illustrating various flow-charting techniques. The computations themselves are of no mathematical importance and there is a simple formula for the n^{th} term, a_n, of this sequence. We derive this formula below. Whether this derivation should be presented to the students as "enrichment" material must be left to the discretion of the teacher. Certainly, if the student can follow the presentation, then the mathematical content (i.e., that of solving recurrence relations) is very important.

If this material is presented to the student, it is likely that he will experience his greatest difficulties in passing from step (3) to step (4) and from step (4) to step (5). In order to clarify the transition from step (3) to step (4), the teacher could copy from the formula of step (3) with k replaced by 2, 3, 4, 5, 6, 7 and then add up the corresponding sides of the equation indicating the sum, with dots (e.g., $a_1 + a_2 + \ldots + a_7$). Now use sigma notation instead of the dots (e.g., $\sum_{k=1}^{7} a_k$) and then replace the 7 by n throughout the formula.

In getting from step (4) to step (5) it should suffice to observe that the indicated sums in step (5) have the same terms as those in step (4). Now we proceed with the derivation of the formula.

The Fibonacci Sequence is determined by the following recurrence relation and initial conditions:

(1) $$a_n = a_{n-1} + a_{n-2} \text{ for } n \geq 2 \text{ ; } a_0 = a_1 = 1.$$

We have seen in the student text how we may successively compute a_2, a_3, a_4, etc. We now derive a formula giving a_n directly without first calculating a_2, a_3, a_4, etc.

First we substitute k for n in formula (1) to obtain

(2) $$a_k = a_{k-1} + a_{k-2} \text{ for } k \geq 2.$$

Next, multiply both sides of this equation by x^k, yielding[†]

$$(3) \qquad a_k x^k = a_{k-1}x^k + a_{k-2}x^k \quad \text{for } k \geq 2.$$

Now sum from $k = 2$ to n,

$$(4) \qquad \sum_{k=2}^{n} a_k x^k = \sum_{k=2}^{n} a_{k-1}x^k + \sum_{k=2}^{n} a_{k-2}x^k .$$

Rewrite the last two sums so as to have a_k appear in the summand

$$(5) \qquad \sum_{k=2}^{n} a_k x^k = \sum_{k=1}^{n-1} a_k x^{k-1} + \sum_{k=0}^{n-2} a_k x^{k+2} .$$

From each sum in (5) pull out two terms so as to have the range of the index k the same in all three,

$$(6) \qquad a_n x^n + a_{n-1}x^{n-1} + \sum_{k=2}^{n-2} a_k x^k$$

$$= a_{n-1}x^n + \sum_{k=2}^{n-2} a_k x^{k+1} + a_1 x^2 + \sum_{k=2}^{n-2} a_k x^{k+2} + a_1 x^3 + a_0 x^2 .$$

Combining some terms:

$$(7) \qquad a_n x^n + a_{n-1}x^{n-1}(1-x) = a_1 x(x+x^2) + a_0 x^2 + \sum_{k=2}^{n-2} a_k (x^{k+2} + x^{k+1} - x^k).$$

Recalling from (1) that $a_0 = a_1 = 1$,

$$(8) \qquad a_n x^n + a_{n-1}x^{n-1}(1-x) = x(x+x^2) + x^2 + \sum_{k=2}^{n-2} a_k x^k (x^2 + x - 1)$$

or

$$(9) \qquad a_n x^n + a_{n-1}x^{n-1}(1-x) = x(x+x^2) + x^2 + (x^2 + x - 1) \sum_{k=2}^{n-2} a_k x^k .$$

[†] By skipping step (3) and just summing (2) from $k = 2$ to n (this amounts to substituting 1 for x in (4)) we have

$$\sum_{k=2}^{n} a_k = \sum_{k=2}^{n} a_{k-1} + \sum_{k=2}^{n} a_{k-2}$$

or

$$a_n + \sum_{k=2}^{n-1} a_k = a_1 + \sum_{k=2}^{n-1} a_k + \sum_{k=0}^{n-2} a_0$$

so that

$$a_n = 1 + \sum_{k=0}^{n-2} a_k .$$

This result can also be obtained by substituting 1 for x in (7) below. It will be discovered by the student when he works problem 3, Exercise 4-1.

Now we will determine the value of x so that $x^2 + x - 1 = 0$, whence the last sum in formula (8) will drop out. Using the quadratic formula, the roots of

$$x^2 + x - 1$$

we found to be

$$\frac{-1 + \sqrt{5}}{2} \quad \text{and} \quad \frac{-1 - \sqrt{5}}{2}$$

For convenience we write $r = \dfrac{\sqrt{5} + 1}{2}$ so that the two roots are

(10) $$\frac{1}{r} = \frac{-1 + \sqrt{5}}{2} \quad \text{and} \quad -r = \frac{-1 - \sqrt{5}}{2}.$$

If x designates either of these roots, we have $x^2 + x = 1$ and $1 - x = x^2$ so that (9) becomes

(11) $$a_n x^n + a_{n-1} x^{n+1} = 1$$

or

(12) $$a_n + a_{n-1} x = \frac{1}{x^n}$$

Substituting each of the roots for x we have the system,

(13) $$\begin{aligned} a_n + a_{n-1} \frac{1}{r} &= r^n \\ a_n - a_{n-1}\, r &= \left(-\frac{1}{r}\right)^n \end{aligned}$$

Now we multiply the first of these equations by r and the second by $\frac{1}{r}$ and add to eliminate a_{n-1}.

(14) $$\begin{aligned} r \qquad a_n + a_{n-1} \frac{1}{r} &= r^n \\ \frac{1}{r} \qquad a_n - a_{n-1}\, r &= \left(-\frac{1}{r}\right)^n \end{aligned}$$

$$\left(r + \frac{1}{r}\right) a_n = r^{n+1} - \left(-\frac{1}{r}\right)^{n+1}$$

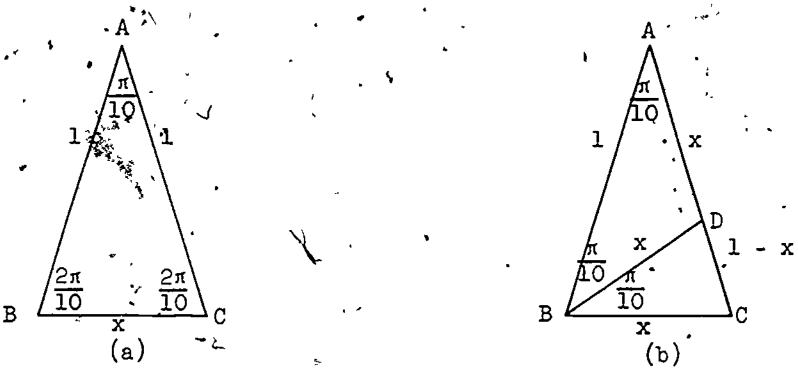
From (10) $r + \frac{1}{r}$ is seen to be simply $\sqrt{5}$ so that

(15) $$a_n = \frac{1}{\sqrt{5}} \left(r^{n+1} - \left(-\frac{1}{r}\right)^{n+1}\right)$$

which is the desired formula.

It is seen that the term $(-\frac{1}{r})^{n+1}$ tends quite rapidly to zero so for large n, a_n is approximately $\frac{1}{\sqrt{5}} r^{n+1}$ where r is approximately 1.618.

It is, further, easy to calculate that $r = \frac{1}{2}\csc\frac{\pi}{20}$ as seen below. In the isosceles triangle (a)



(a) (b)

we draw from the vertex B a segment BD bisecting $\angle$ ABC to obtain figure (b). From the similarity of $\triangle$ ABC and $\triangle$BCD we have

$$\frac{x}{1-x} = \frac{1}{x}$$

or

$$x^2 + x - 1 = 0$$

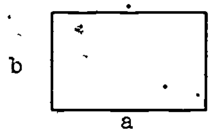
so that

$$x = \frac{-1+\sqrt{5}}{2} = \frac{1}{r} .$$

But, from the isosceles triangle in (b) we see that $\frac{x}{2} = \sin\frac{\pi}{20}$. Thus, $r = \frac{1}{2}\csc\frac{\pi}{20}$ and (15) becomes

(16) $$a_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1}{2}\csc\frac{\pi}{20}\right)^{n+1} - \left(-2\sin\frac{\pi}{20}\right)^{n+1}\right).$$

A last interesting point concerning the Fibonacci sequence involves the "golden mean". Now you may know that a rectangle is said to have



the golden proportion if

(17) $$\frac{a+b}{a} = \frac{a}{b}$$

Letting $\frac{a}{b} = x$, then (17) becomes

$$1 + \frac{1}{x} = x \quad \text{or} \quad x^2 - x - 1 = 0 .$$

so that x (necessarily positive) is

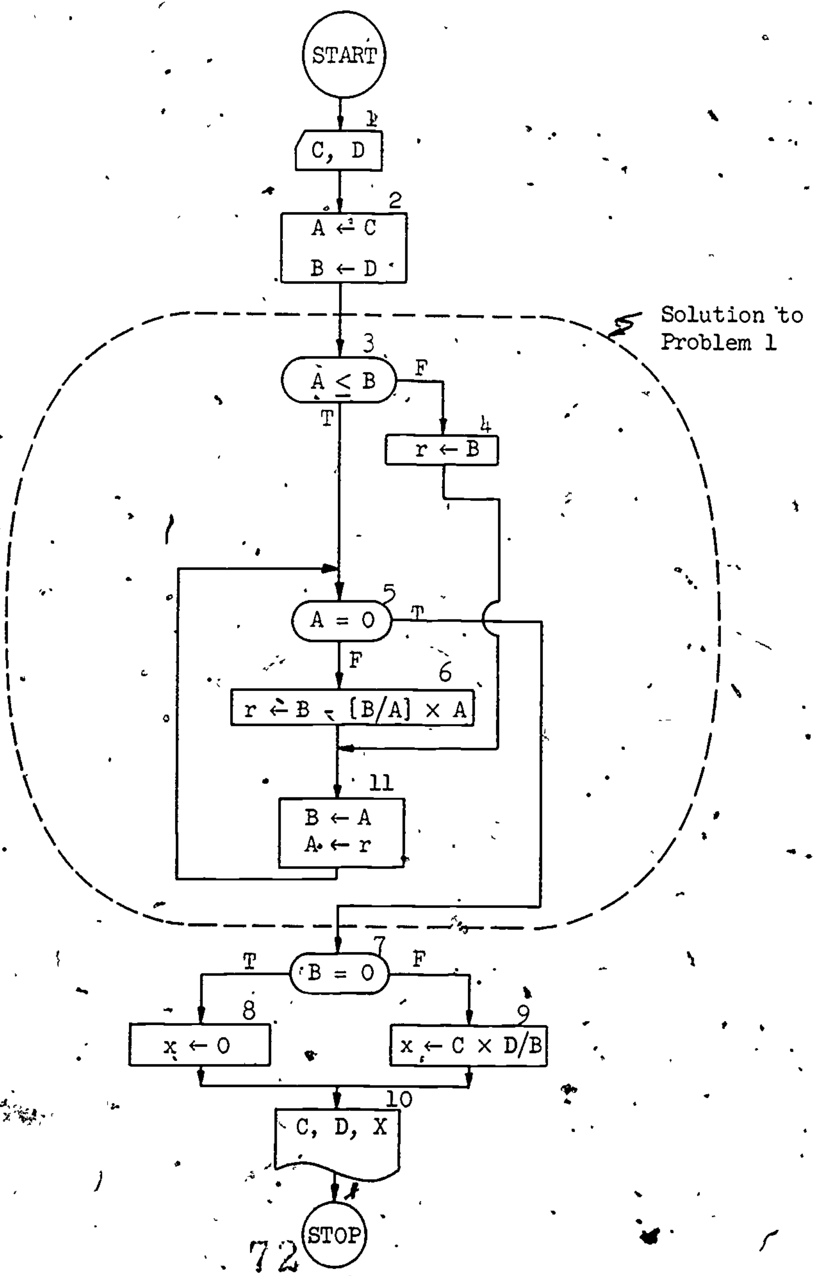
$$\frac{1+\sqrt{5}}{2} \quad \text{or} \quad r .$$

The Fibonacci Sequence is of little mathematical importance in itself. However, as has been seen, it is pedagogically quite useful in introducing some interesting mathematical techniques.

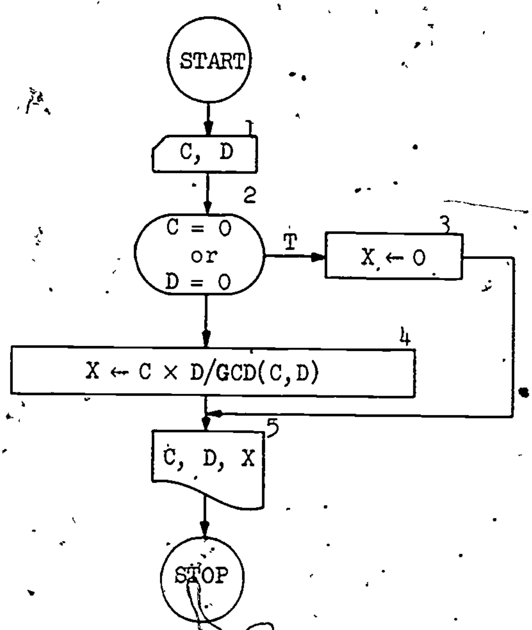
Answers to Exercises 3-2 Set B

1. What is required is a simple change to take advantage of the fact that the last two assignments of box 4 occur again in box 6. The necessary change is incorporated in the flow chart for the next exercise.

2. The statement in the hint is justified as follows: If X is the g.c.d. of C and D, then there are integers m and n so that $C = mX$ and $D = nX$. Moreover, m and n have no factors in common. Thus, $m \times n \times X$ is the smallest integer which is a multiple of both mX and nX. Now, $m \times n \times X = mX \times nX/X = C \times D/X$.



Comment: After functional procedures have been studied in Section 5-3, we will be able to simplify the flow chart to:



Additional remarks on GCD algorithms

Here we present a more difficult algorithm which the teacher may wish to use as a project for better students.

In connection with the Euclidean Algorithm there is an important mathematical theorem to the effect that, given non-negative integers A and B, there are integers X and Y so that GCD(A,B) = AX + BY. The problem of finding the values of X and Y involves some mathematical preparation. Letting a_0, a_1 be the given numbers whose GCD we are to find, we consider a sequence

$$a_0, a_1, a_2, a_3, \ldots, a_n$$

where for each $k > 1$, a_k is the remainder on dividing a_{k-2} by a_{k-1}. The formula $r = a - qb$ becomes

(1) $$a_k = a_{k-2} - q_k a_{k-1} = c_k a_0 + d_k a_1 .$$

That these c_k and d_k really exist can be seen by successive substitution and verified inductively. When $a_k = \text{GCD}(A,B)$, then $c_k = X$ and $d_k = Y$. These last c_k and d_k are the only ones we wish to output.

Clearly,

$$c_0 = 1, \quad d_0 = 0, \quad c_1 = 0, \quad d_1 = 1.$$

In order to derive the recurrence relation for the c_k and d_k we write out (1) for three values of k,

$$a_{k-2} = c_{k-2}a_0 + d_{k-2}a_1$$

$$a_{k-1} = c_{k-1}a_0 + d_{k-1}a_1$$

$$a_k = a_{k-2} - q_k a_{k-1}$$

Substituting the first two lines into the third

$$a_k = c_{k-2}a_0 + d_{k-2}a_1 - q_k(c_{k-1}a_0 + d_{k-1}a_1).$$

Rearranging terms

$$a_k = (c_{k-2} - q_k c_{k-1})a_0 + (d_{k-2} - q_k d_{k-1})a_1.$$

Thus,

$$c_k = c_{k-2} - q_k c_{k-1}, \quad d_k = d_{k-2} - q_k d_{k-1}.$$

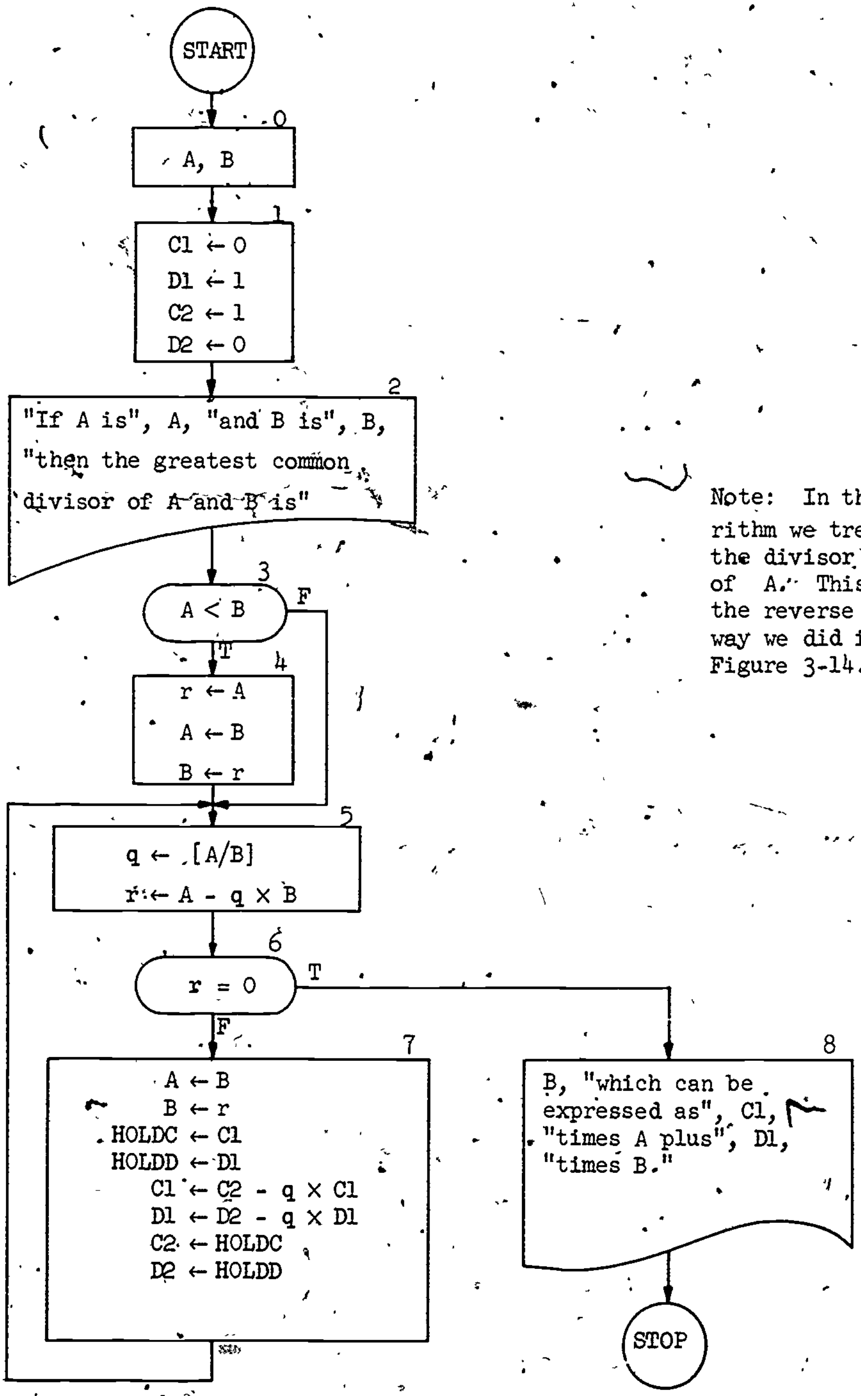
We now see that only the last two c's and the last two d's are needed. Letting the latest c_k be C1 and the next to last be C2 (similarly for the d_k) we have the assignments:

```
HOLDC ← C1
C1 ← C2 - q × C1
C2 ← HOLDC
```

Here q denotes q_k. A complete flow chart is given below.

Flow chart for computing GCD of A + B and representing it as

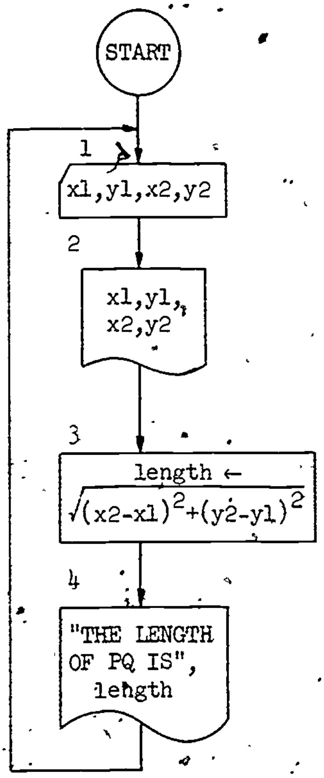
$$GCD(A,B) = AX + BY$$



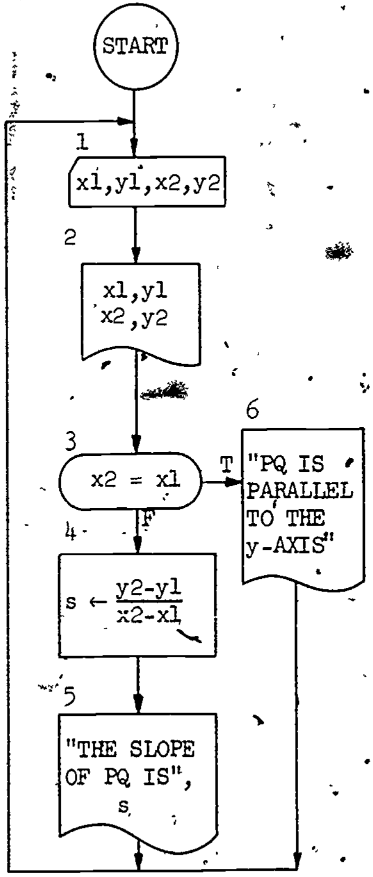
Note: In this algorithm we treat B as the divisor instead of A. This is just the reverse of the way we did it in Figure 3-14.

Answers to Exercises 3-2, Set C

1.



2.



1.

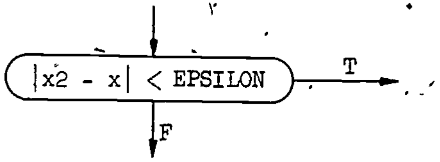
Test Data	Input x1, y1, x2, y2	Output (box 4)
1	-3, 2 , 1, 5	The length of PQ is 5.0

2.

Test Data	Input x1, y1, x2, y2	Output (boxes 5 or 6)
1	4 , 2 , 3, 5	The slope of PQ is -3.0
2	2 , 5 , 2, 6	PQ is parallel to the y-axis.

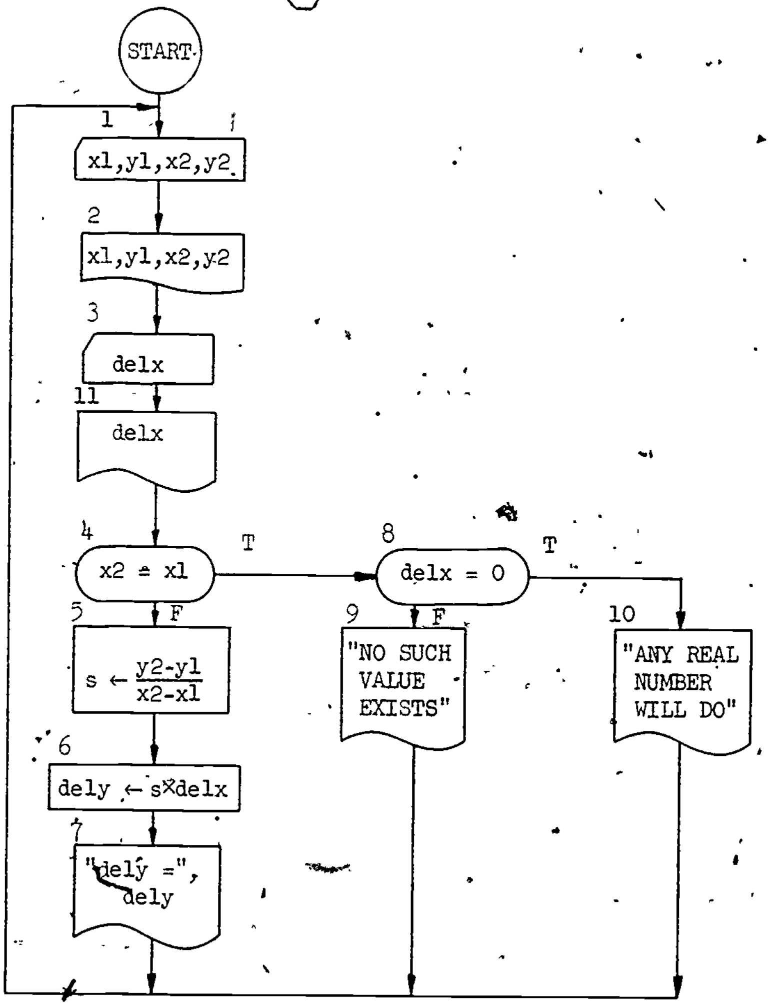
Comment on 2

In realistic problems one rarely seeks a check for true equality of two real numbers (box 3). When the numbers being compared are measured data or computed values, some uncertainty is associated with each value. Instead, tests like



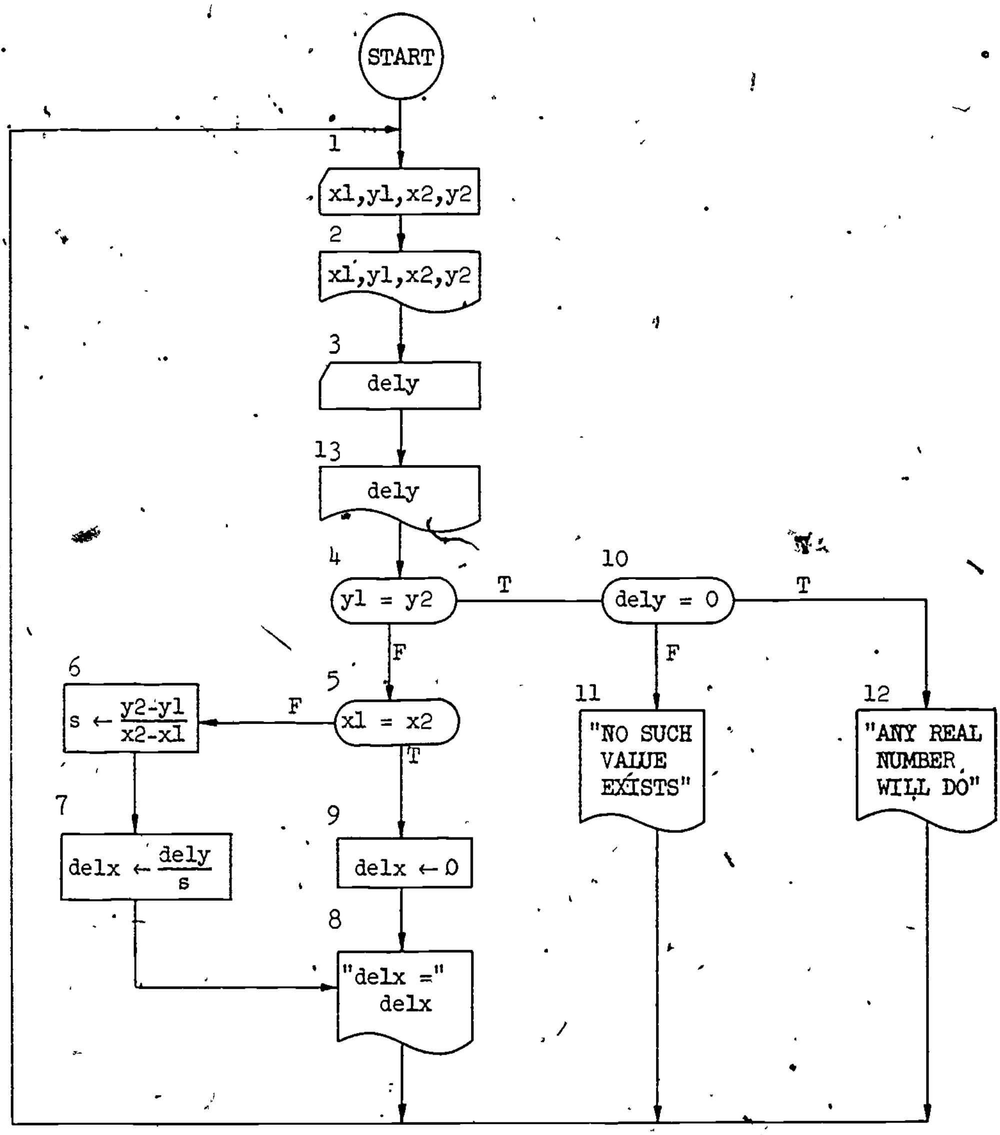
are used, where epsilon is some small value determined by the problem analyst or programmer. We shall make use of these ideas in Chapter 7.

3.



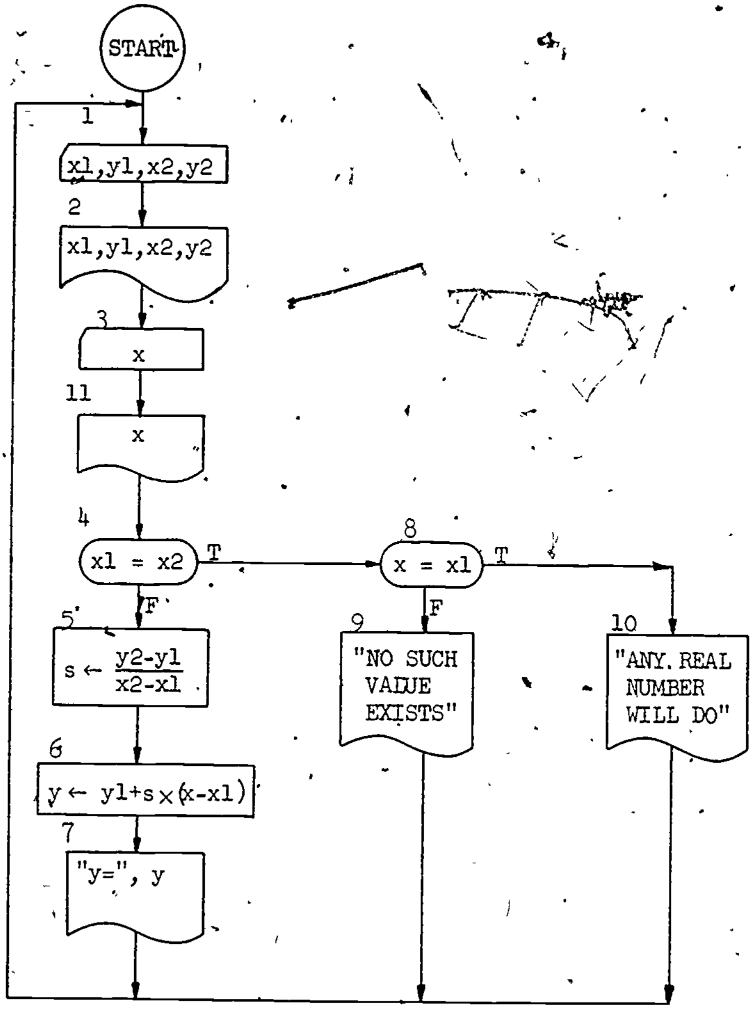
Test Data	Input x1, y1, x2, y2	delx	Result output (box 7 or 9 or 10)
1	4, 2, 3, 5,	.1	dely = -.3
2	4, 2, 4, 5,	.1	No such value exists
3	4, 2, 4, 5,	.0	Any real number will do

4.



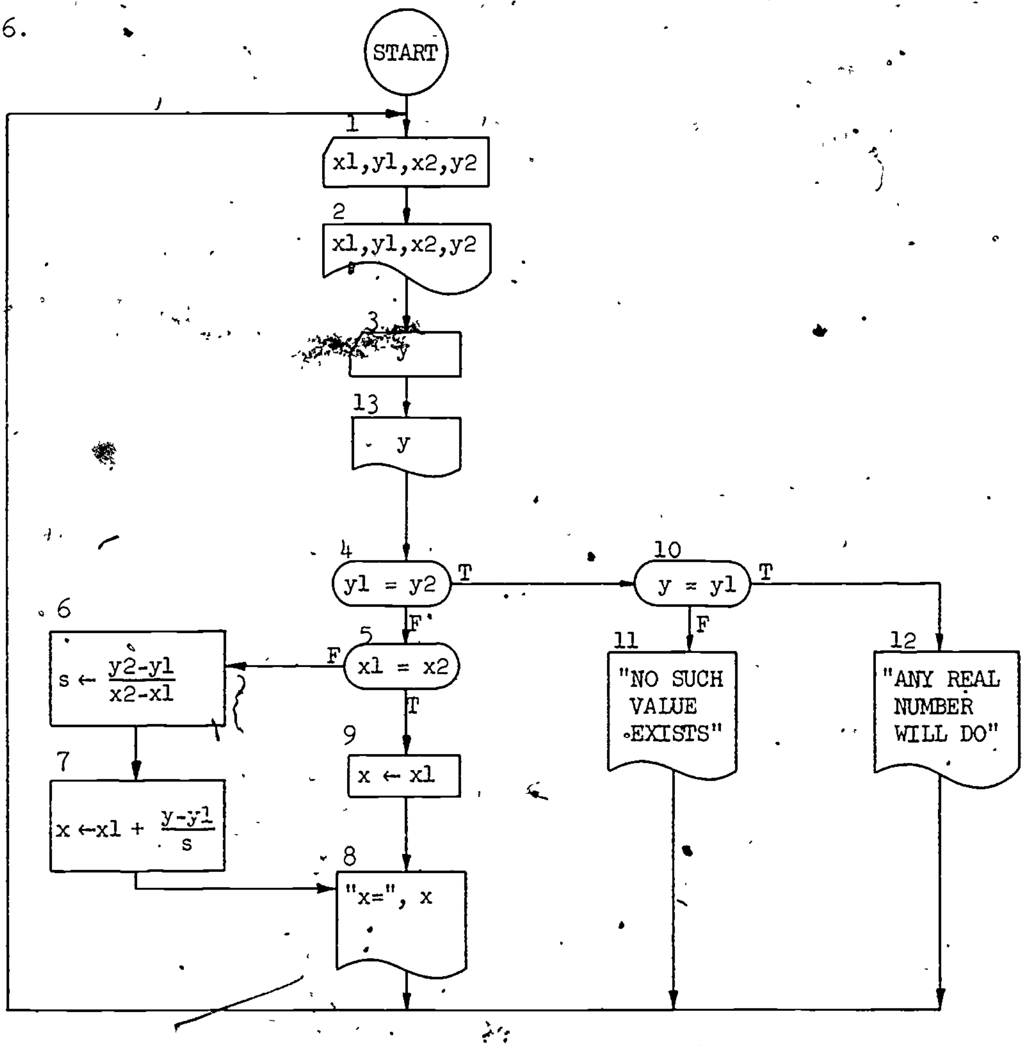
Test Data	Input xl, yl, x2, y2, dely	Result Output (boxes 8, 11, or 12)
1	4, 2, 3, 5, 0.3	delx = -0.1
2	4, 5, 3, 5, 0.3	No such value exists
3	4, 2; 4, 5, 0.3	delx = 0.0
4	4, 5, 3, 5, 0.0	Any real number will do

5.



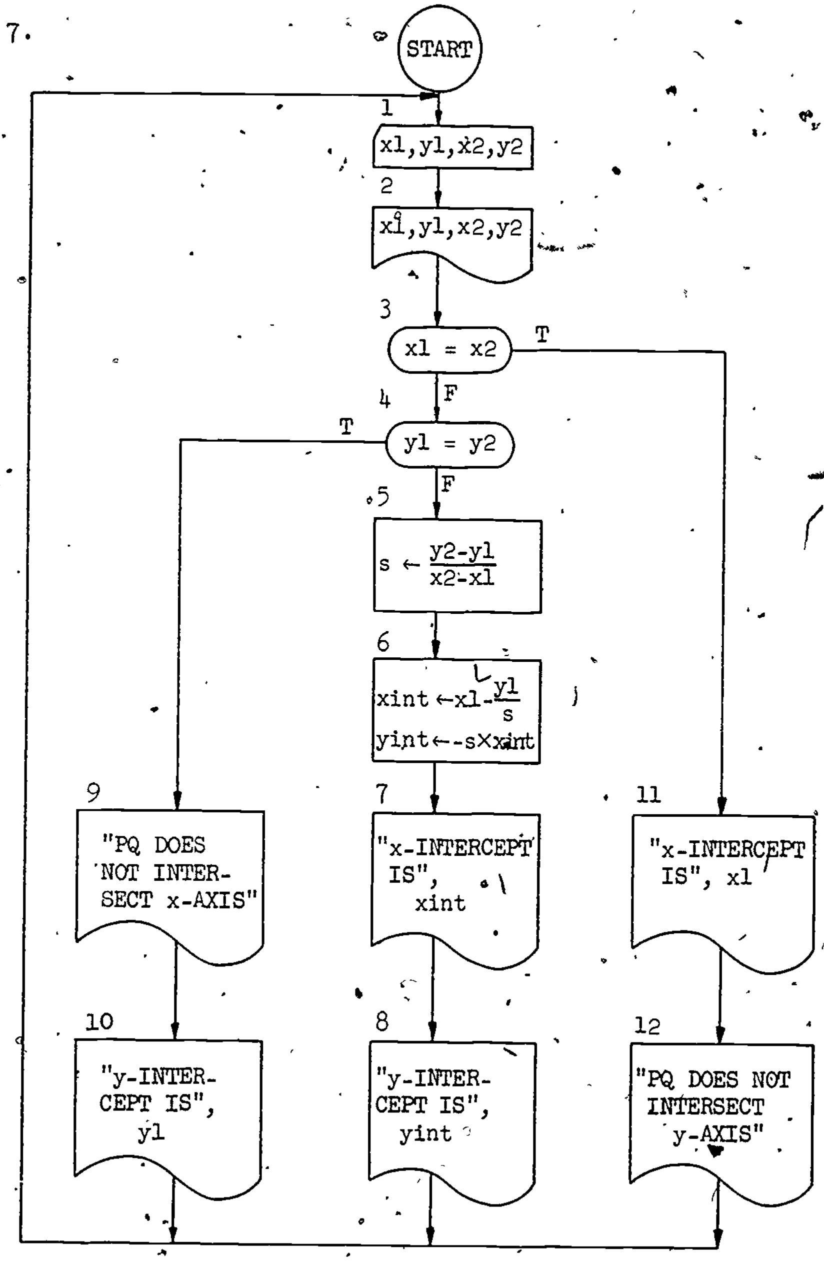
Test Data	Input x1, y1, x2, y2, x	Result output (boxes 7, 9, or 10)
1	4, 2, 3, 5, 7	y = -7
2	4, 2, 4, 5, 4	Any real number will do
3	4, 2, 4, 5, 7	No such value exists

6.

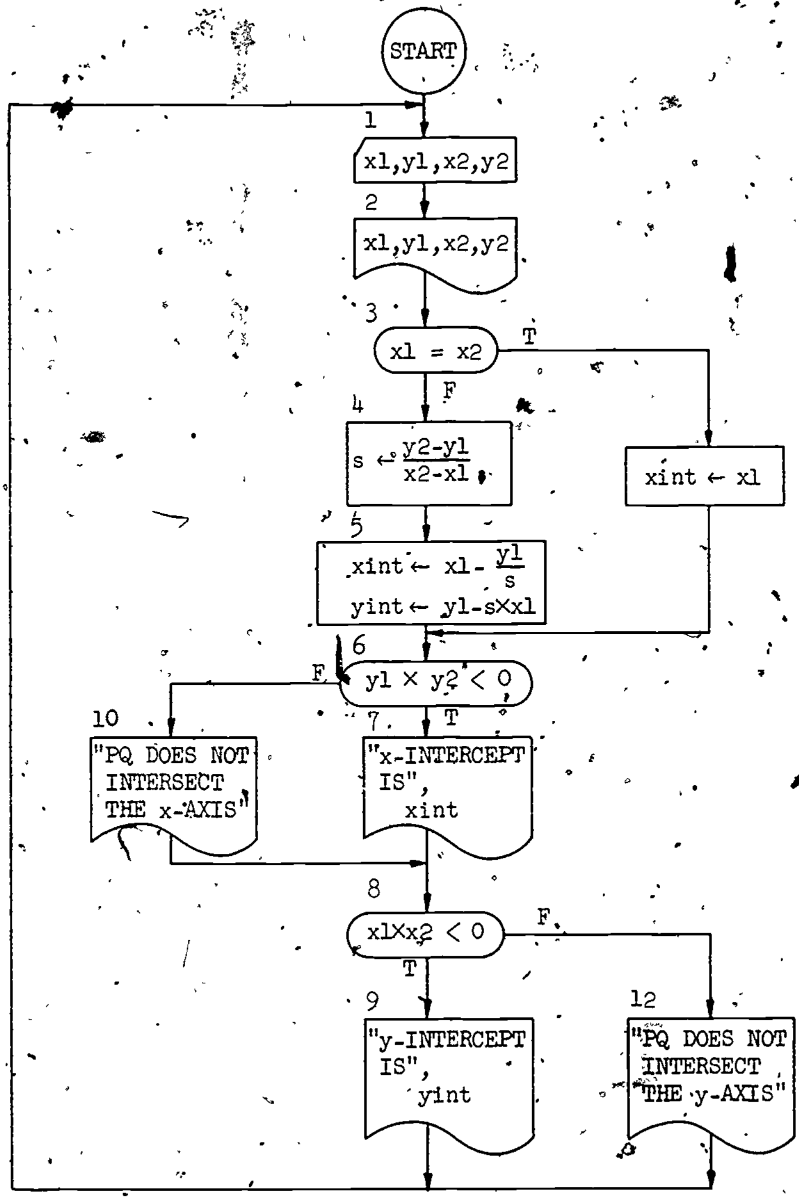


	Input					Result output (boxes 8, 11, or 12)
Test Data	x1,	y1,	x2,	y2,	y	
1	4,	2,	3,	5,	-4.0	x = 6.0
2	4,	2,	3,	2,	-4.0	No such value exists
3	4,	2,	3,	2,	2	Any real number will do
4	4,	2,	4,	2,	4.0	No such value exists

7.



Test Data	Input x1, y1, x2, y2	Result output (boxes 9,10,or 6,7,or 11,12)
1	4, 2, 3, 5	x-intercept is 4.67 y-intercept is 14.0
2	4, 2, 4, 5	x-intercept is 4 PQ does not intersect y-axis
3	4, 2, 3, 2	PQ does not intersect x-axis y-intercept is 2.0



Test Data	Input	Result output (boxes 10,6,8,11,12)
	x1, y1, x2, y2	
1	4, 2, 3, 5	PQ does not intersect the x-axis PQ does not intersect the y-axis
2	-4, -2, 3, 5	x-intercept is -2.0 y-intercept is 2.0
3	-4, 2, 3, 5	PQ does not intersect the x-axis y-intercept is 3.71
4	4, -2, 3, 5	x-intercept is 3.71 PQ does not intercept the y-axis
5	4, -2, 4, 5	x-intercept is 4.0 PQ does not intercept the y-axis

Exercises 3-3

Comment on Exercises 1 - 7

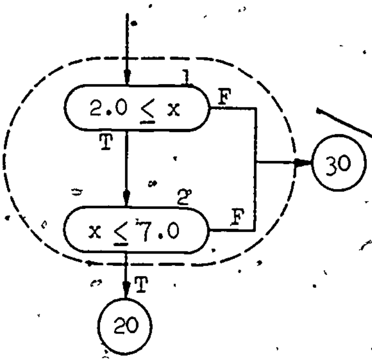
We regard this as another key exercise set because of the practice to be gained in synthesizing the flow chart from verbal statements of fairly complicated conditionals. As many of these as possible should be worked. Exercises like these would be excellent for tests. They are easy to make up. A minimum subset to be assigned would be 1, 3, 4, 5 or 6, and possibly 7. Number 7 should not be assigned unless it is preceded by 4. The companion exercises in the language manual should also be assigned.

Comment on Exercises 8 and 9

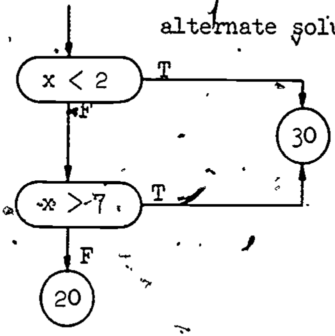
These are the reverse of Exercises 1 through 7. The student is now challenged to take a flow chart of a compound condition and come up with a corresponding geometric region determined by one of the two exits from this condition. At least one of these exercises should be assigned, and a test question of this kind is recommended. They are easy to invent.

Answers to Exercises 1-7 of Section 3-3

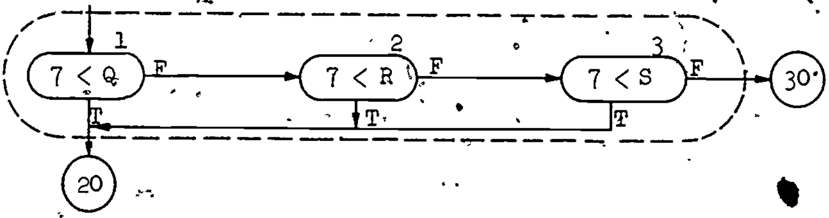
1.



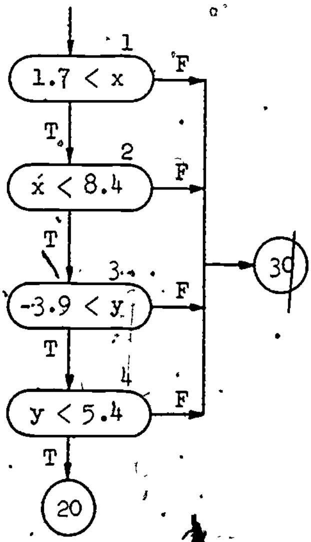
alternate solution



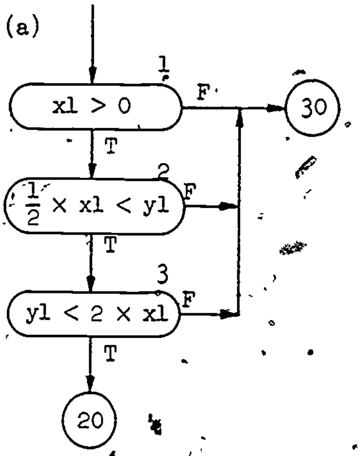
2.



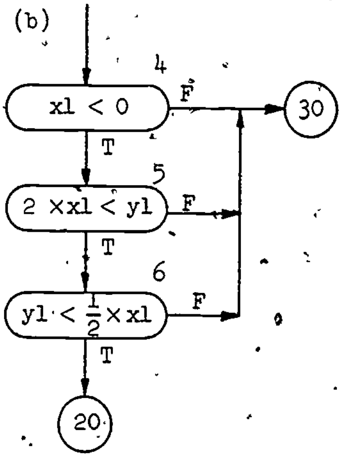
3.



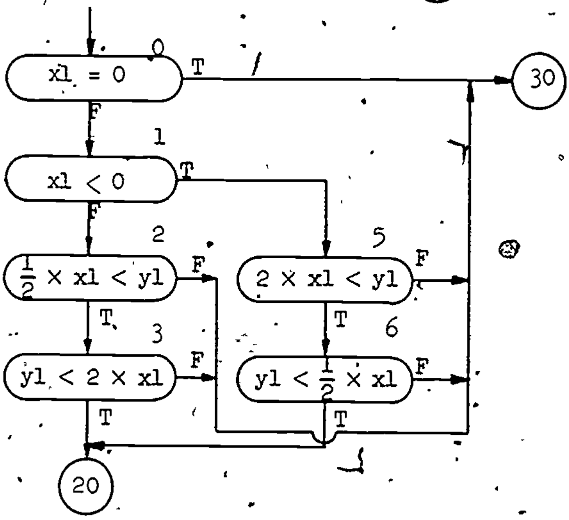
4. (a)


1
xl > 0
F
30
T
2
$\frac{1}{2}$ × xl < yl
F
T
3
yl < 2 × xl
F
T
20

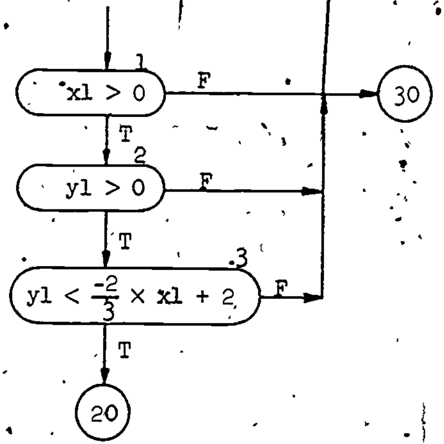
(b)


4
xl < 0
F
30
T
5
2 ×xl < yl
F
T
6
yl < $\frac{1}{2}$×xl
F
T
20

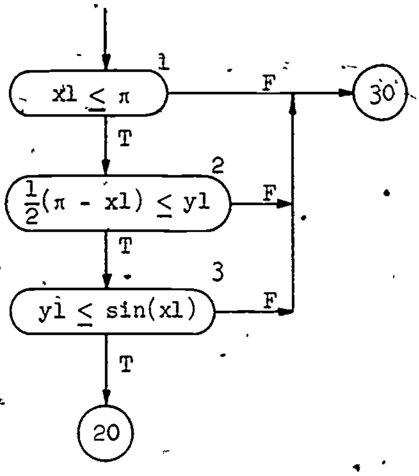
(c)


0
xl = 0
T
30
F
1
xl < 0
T
F
2
$\frac{1}{2}$ × xl < yl
F
T
3
yl < 2 × xl
F
T
5
2 × xl < yl
F
T
6
yl < $\frac{1}{2}$ × xl
F
T
20

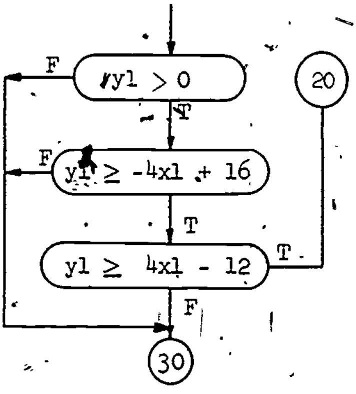
5.

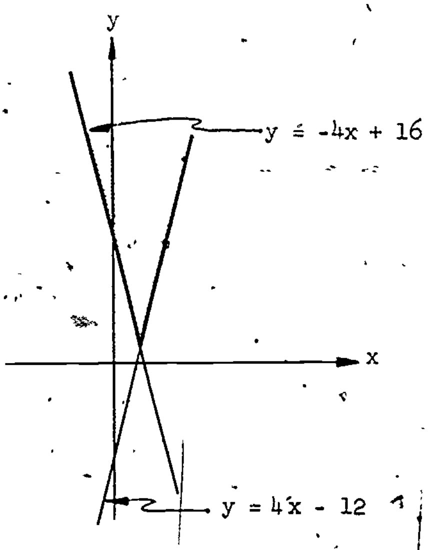

1
xl > 0
F
30
T
2
yl > 0
F
T
3
yl < $\frac{-2}{3}$ × xl + 2
F
T
20

6.

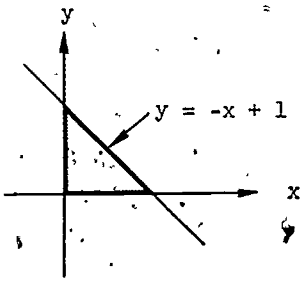

1
$x1 \leq \pi$
F
30
T
2
$\frac{1}{2}(\pi - x1) \leq y1$
F
T
3
$y1 \leq \sin(x1)$
F
T
20

7.

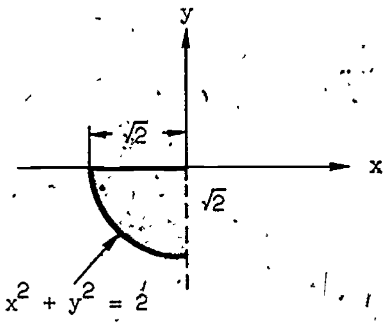

F
$y1 > 0$
T
20
F
$y1 \geq -4x1 + 16$
T
$y1 \geq 4x1 - 12$
T
F
30


y
$y = -4x + 16$
x
$y = 4x - 12$

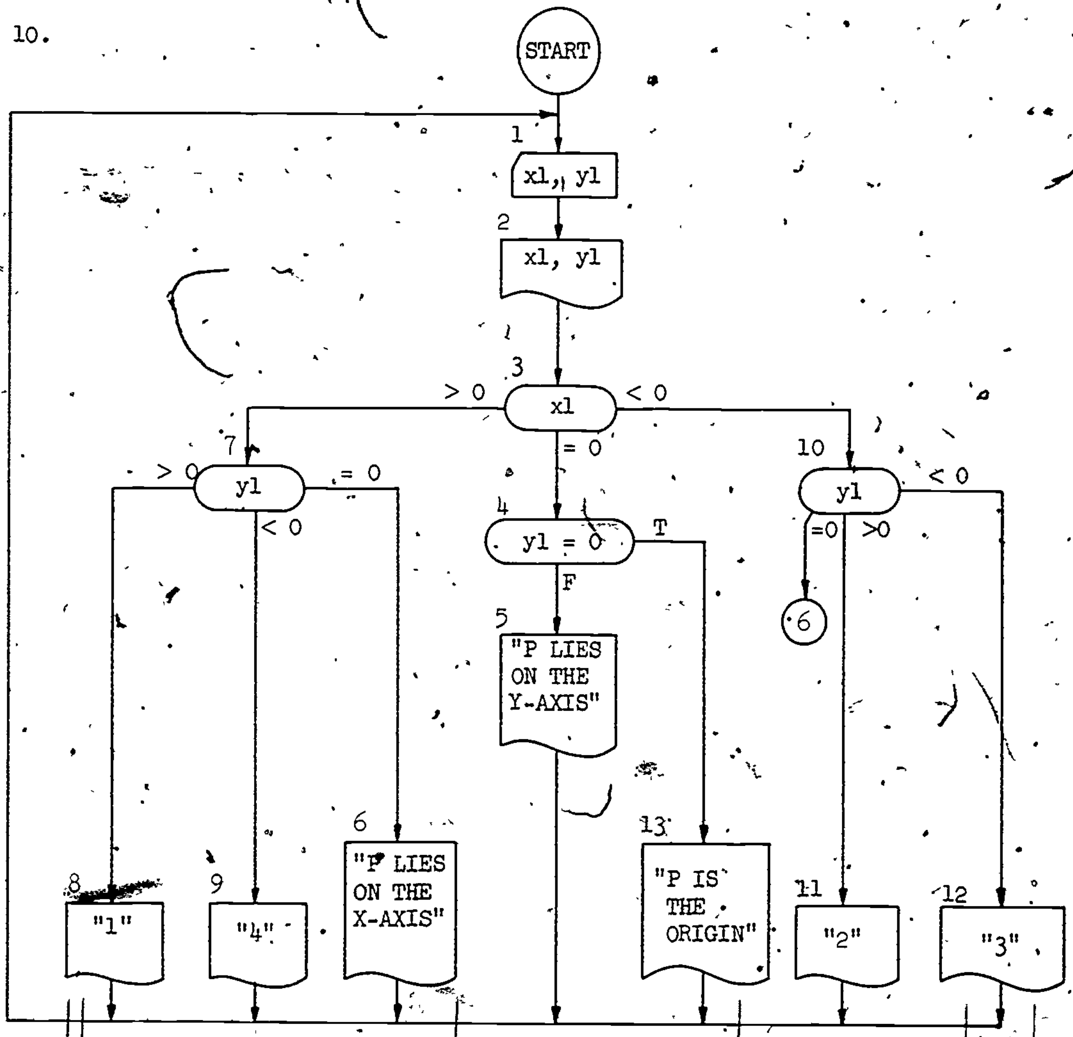
8.


y
$y = -x + 1$
x

9.

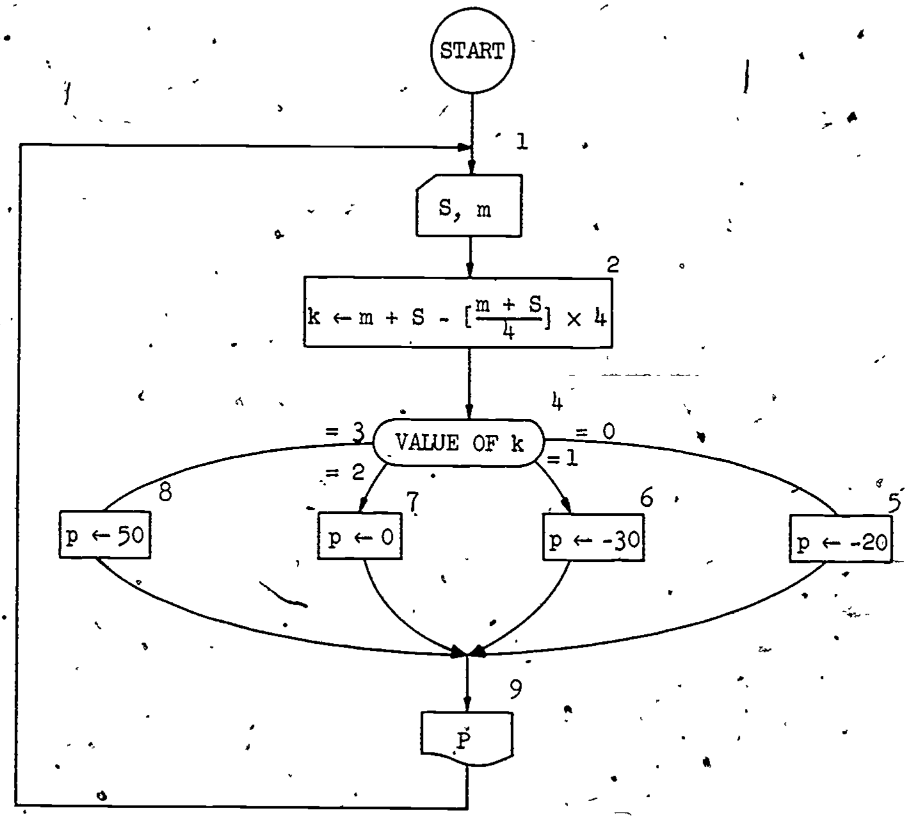

y
$\sqrt{2}$
x
$\sqrt{2}$
$x^2 + y^2 = 2$

10.



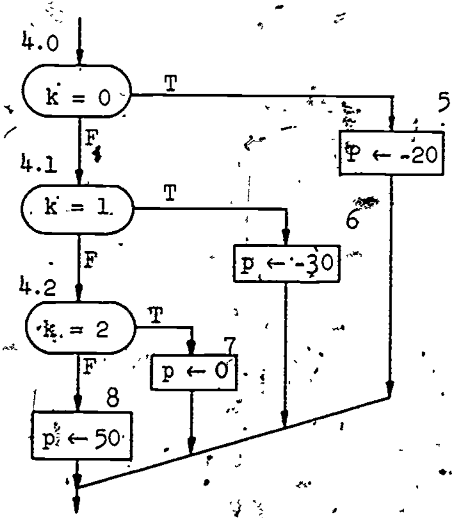
Test Data	Input xl, yl	Result output (from boxes 4,6,8,9,11,12,13)
1	0, 2,	P lies on the y-axis
2	0, 0	P lies on the y-axis P is the origin
3	1, 0	P lies on the x-axis
4	1, 3	1
5	1, -3	4
6	-1, 3	2
7	-1, -3	3

11.



Comment:

Boxes 4, 5, 6 and 7 can be alternately represented as:



Illustrating that a four-way branch may be decomposed into three 2-way branches. (In general, an N-way branch may be decomposed into a chain of N - 1 2-way branches.)

Test Data	Input S M	Output (box 9) I
1	21, 165	0
2	24, 169	-30
3	23, 161	-20
4	27, 188	50

This is the second of these problems related to the carnival wheel. The third problem occurs in Section 3-5 where we show the student how the conditional can be avoided using a vector whose four elements comprise the scoring rule.

Answers to Exercises 3-5 Set A

1. The student's claim is correct. (b) is equivalent, simpler and more general. If we presume the data values for P_1, P_2, P_3, and P_4 which are input at box 0 are 50, 0, -30, and -20, respectively, then the two flow charts are equivalent. Instead of k being used as a test value in a conditional box, it is used as in the subscript expression

$$k + 1$$

Thus, if k were 3, P_{k+1} means P_4. If for P_4 we had assigned -20, the same value for P_{k+1} is printed in (b) as is printed for P in the corresponding box 9 of flow chart (a). The same match in values printed can be seen to hold for the other three values which can be computed for k.

2. Any 4-way point rule can now be devised using the algorithm in (b) by merely repeating the execution of the algorithm with different data values for P_1, P_2, P_3, and P_4. This is not true for flow chart (a). In order to change the point rule we must change the values assigned to P in boxes 5, 6, 7 and/or 8.

Comment

Flow chart (b) represents a proper use of subscripts in a computer. In the next group of exercises we see an example of a poor use of subscripts.

3. 100 times.

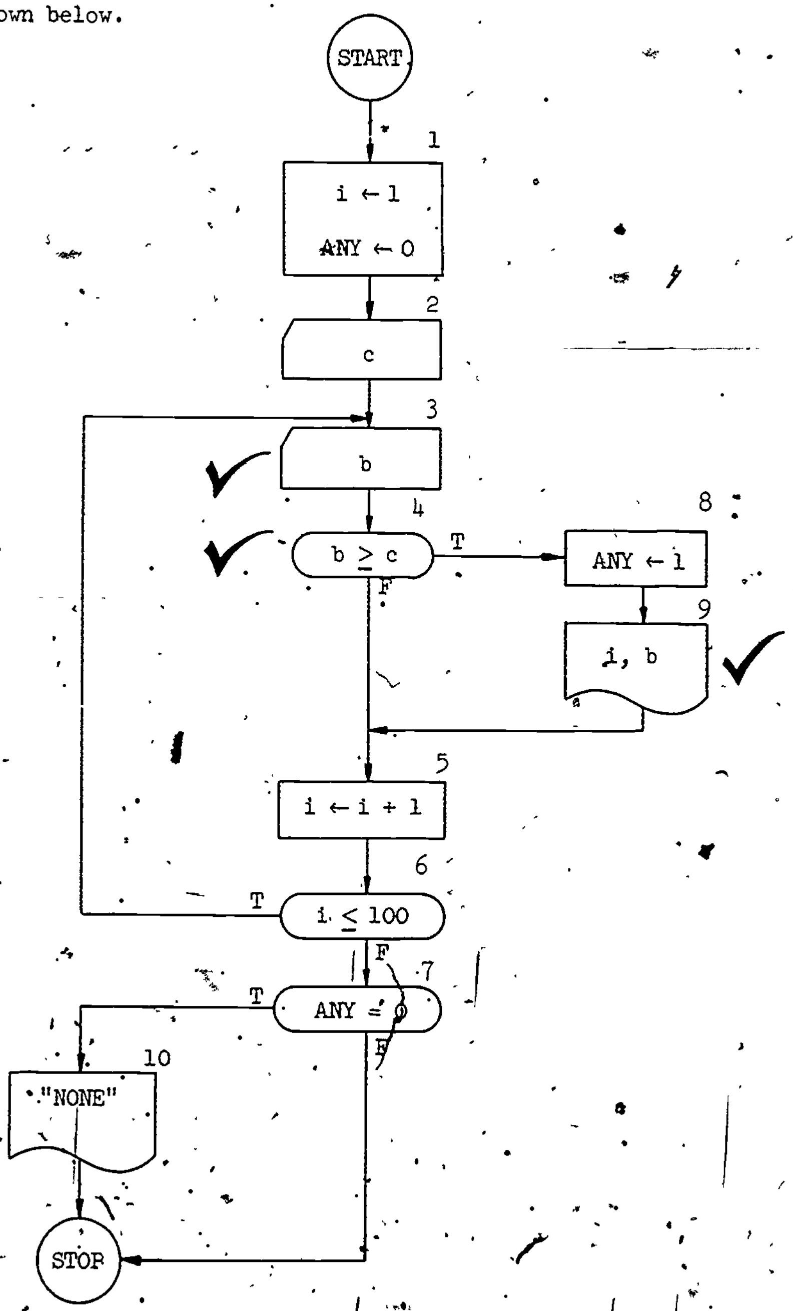
4. Once for each time a value of b is encountered which is greater than or equal to m. It need not happen at all, or it can happen a hundred times.

5. If box 8 is never executed, then box 10 will be executed. If box 8 is executed even once, then box 10 will not be executed. In other words, ANY is made to behave like a two-way switch. When box 7 is reached, the value of ANY is tested. Its value is either equal to 0 or to 1, depending on earlier events. It is set initially to zero in box 3 and may or may not be reset to a value of one in box 8. It makes no difference how many times box 8 is executed after it happens once, its value remains one.

Comment: Switch variables are useful for recalling whether or not a certain earlier event has occurred. In general, whenever a certain event occurs, like the passing of "Train No. 9", the switch is set to an alternate value, say 1, thus changing its initially chosen value. Now, if ever we need to know if that certain event occurred--even once--all we have to do is determine the current value of the switch by testing to see if it is one or zero.

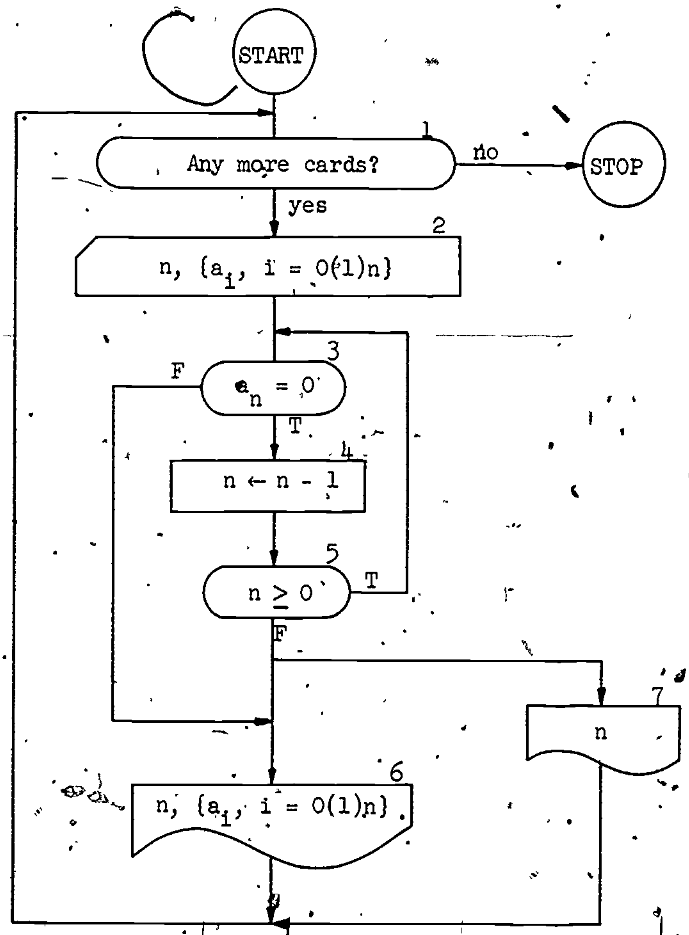
6. No. It is not necessary for more than one value to be in memory at any one time.

The modified flow chart, before generalizing to read in n values, is shown below.



7. We could read the value of n as part of box 2 of Figure 3-25. Then, in boxes 3 and 6, we must replace 100 by n.

8. This problem will reappear as a procedure in Section 5-5.



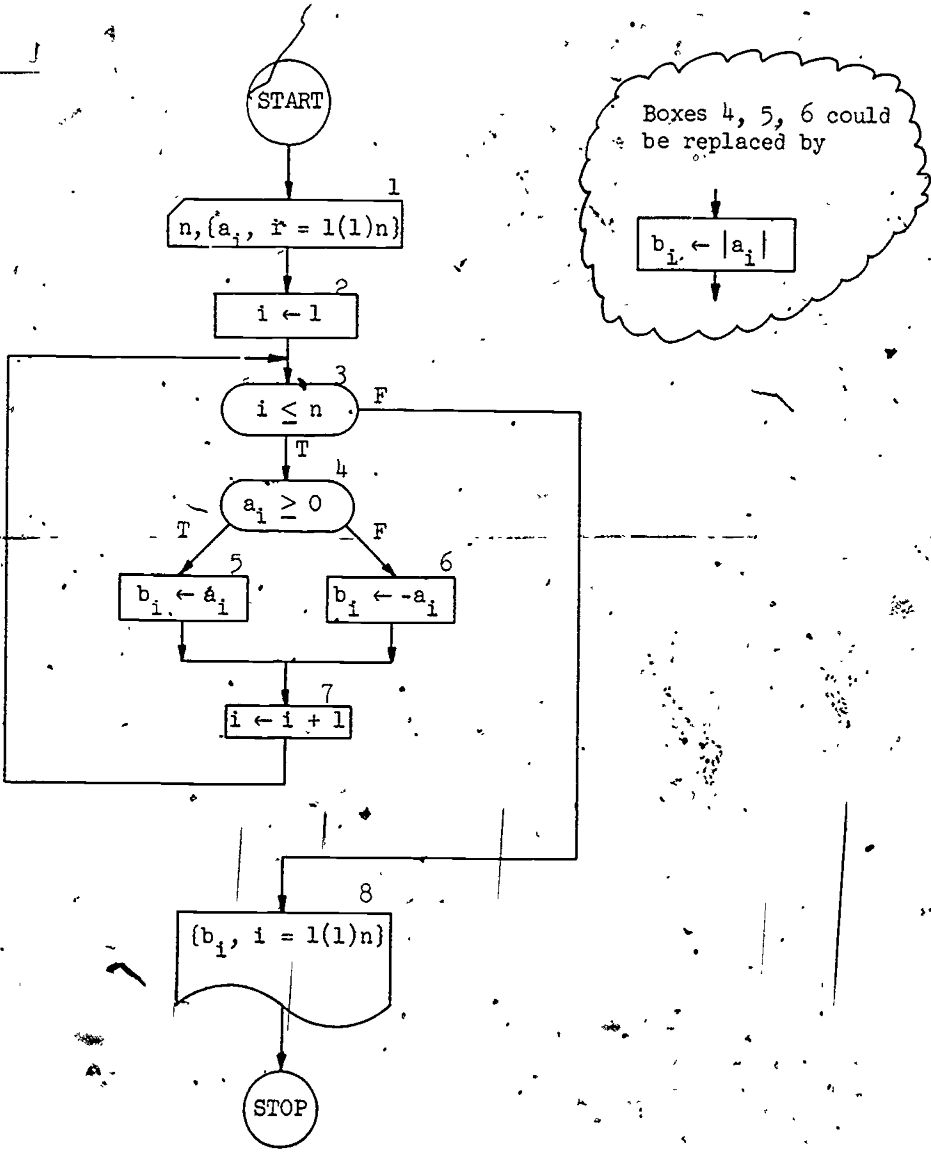
Note: Strictly speaking, box 7 is not necessary. That is, the false exit from box 5 could be led into box 6. When exiting "false" at box 5, $n = -1$. In this event, if we enter box 6, the value of n would be printed but none of the a's, because for $i = 0$, i already exceeds n, so the set is empty.

We have separated boxes 6 and 7 in anticipation of the difficulty which we would encounter in some FORTRANs where using the equivalent "implied DO loop notation", the first element is always taken because the test $(i > n)$ is not made until after the first item (first transit through the implied loop) is printed.

Here is a very simple question question and its flow chart solution:

Draw a flow chart for inputting a set of values and outputting another set having as its elements the absolute values of the elements of the input set.

Solution:



Note: In Section 4-1 the student could be asked to redo this flow chart pulling boxes 2, 3 and 7 together into an iteration box.

Answers to Exercises 3-5, Set B

1. (a) 4 , 7 , 2 , -5 , 4
n

(b)

box sequence	3	4	5	6	7	8	Current Value of K
1	✓						1
2		✓					
3			✓				
4	✓						1
5		✓					
6				✓			2
7					✓		
8		✓					
9			✓				
10	✓						1
11		✓					
12			✓				
13	✓						1
14		✓					
15				✓			2
16					✓		
17		✓					
18				✓			3
19					✓		
20		✓					
21			✓				
22	✓						1
23		✓					
24				✓			2
25					✓		
26		✓					
27				✓			3
28					✓		
29		✓					
30				✓			4
31					✓		
32						✓	
Total	5	10	4	6	6	1	

This much of the table is given to the student.

Appearance of Scratch pad memory for the A vector.

	A			
1	~~7~~	~~2~~	-5	
2	~~2~~	~~7~~	~~-5~~	2
3	~~-5~~	~~7~~	4	
4	~~4~~	7		

(c) 32 boxes

(d) 10 times

2. 3 times or N - 1

3. 13 times

Comment: One popular way to rate sorting algorithms is by the number of comparisons like box 3 which are required as a function of N. In this algorithm the number of comparisons depends strongly on the initial ordering of the data ranging from a minimum of N - 1 when the data are in perfect order up to a maximum of $N^3 + 5N - 6$ when data are initially in reverse order. In Chapter 4 we show an improvement in the algorithm which reduces this maximum to $N \times (N - 1)/2$. Thus, for reverse ordered data we get

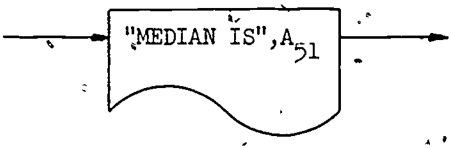
N	Number of comparisons
2	2
3	6
4	13
5	24
6	46

In Chapter 4 we shall develop an algorithm where the number of comparisons is independent of the initial ordering of the data and is equal to $N \times (N - 1)/2$.

This exercise is inspired out of the need to have the student focus his attention, while developing an algorithm, on the question of whether subscripted variables are needed. Certainly none are needed in answering part (a) and (b). At other times they definitely are needed, as in part (d).

Answers to Exercises 3-5 Set C

(a) We can use the Sort flow chart of Figure 3-27 to solve this problem. In box 2, N could be specialized to 101 and in box 8 the output could be changed to

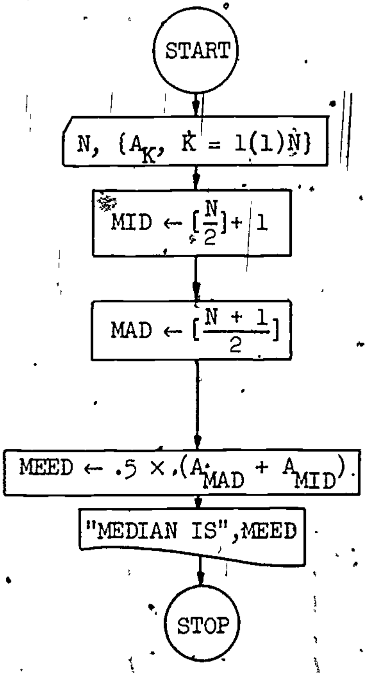


Subscripted variables are essential for an internal sorting algorithm like the one in Figure 3-27. In other words, it is necessary to input all the ages into a vector in memory and then sort.

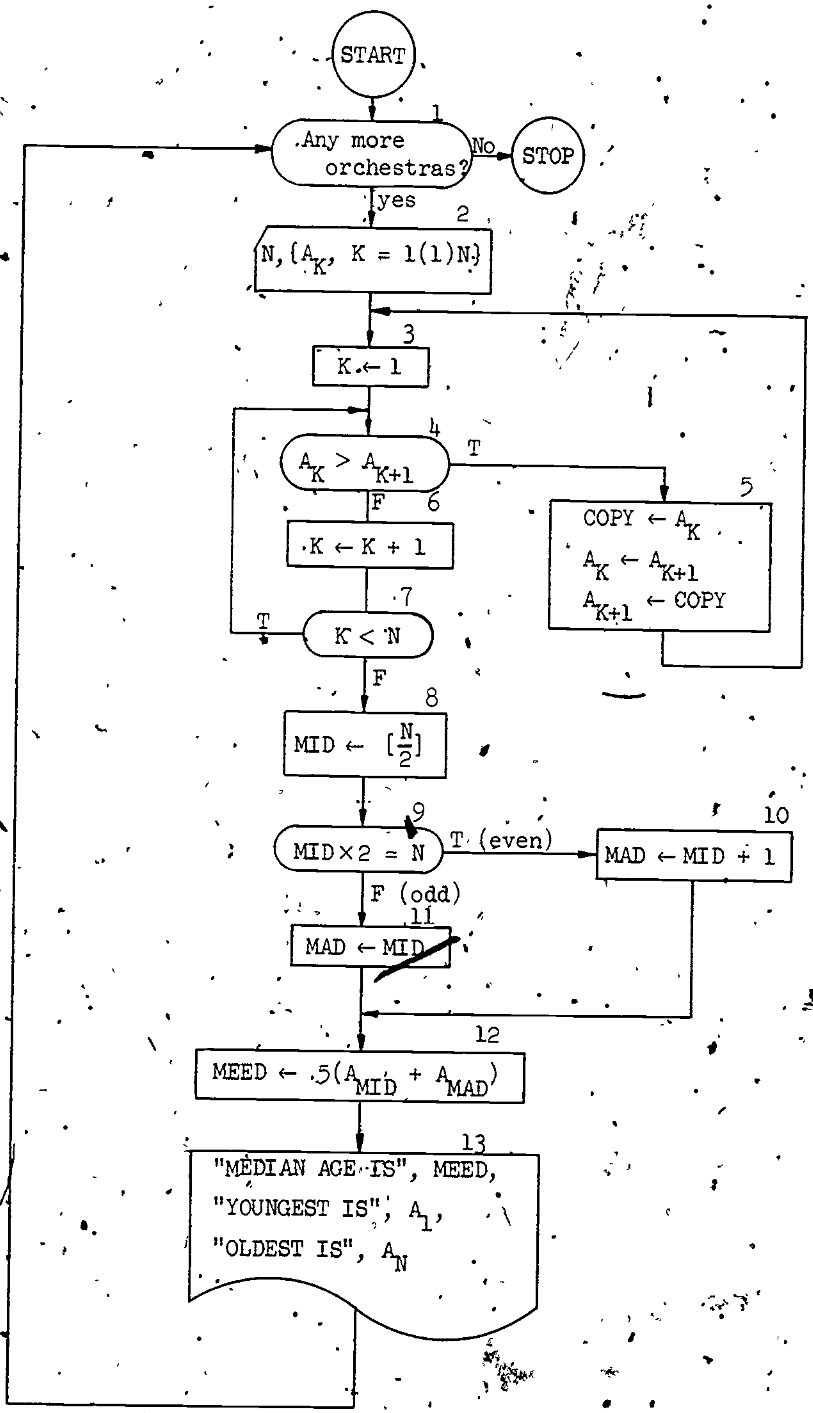
(b) The median of a set of N numbers, N even or odd, can be defined as the average of two numbers according to this formula:

$$\text{MEED} = .5 \times (A_{\left[\frac{N+1}{2}\right]} + A_{\left[\frac{N}{2}+1\right]})$$

If N is odd, $\left[\frac{N+1}{2}\right]$ and $\left[\frac{N}{2}+1\right]$ are equal and both would give the same value as the variable we called $\text{MID} = (\frac{N+1}{2})$. But, if N is even, then $\frac{N}{2}$ is integral and $\left[\frac{N}{2}+1\right]$ reaches the next integer after $\left[\frac{N+1}{2}\right]$. Using this expression the flow chart can be rewritten



(c)

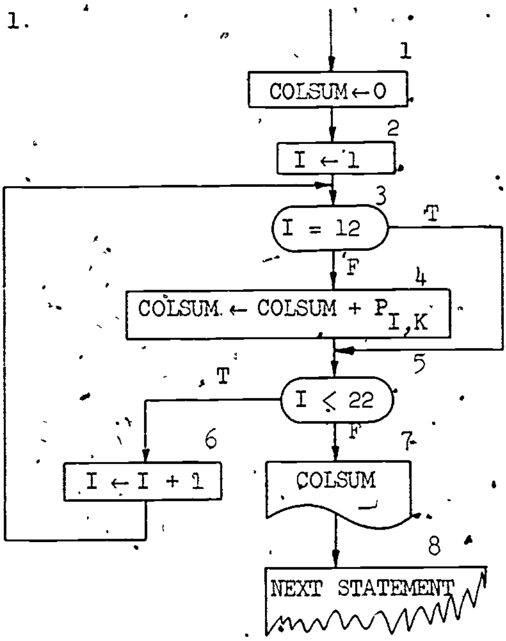

START
1
Any more orchestras?
No
STOP
yes
2
N, {A_K, K = 1(1)N}
3
K ← 1
4
$A_K > A_{K+1}$
T
5
COPY ← A_K
A_K ← A_{K+1}
A_{K+1} ← COPY
F
6
K ← K + 1
7
T
K < N
F
8
MID ← [$\frac{N}{2}$]
9
MID × 2 = N
T (even)
10
MAD ← MID + 1
F (odd)
11
MAD ← MID
12
MEED ← .5(A_{MID} + A_{MAD})
13
"MEDIAN AGE IS", MEED,
"YOUNGEST IS", A_1,
"OLDEST IS", A_N

Exercises 3-6

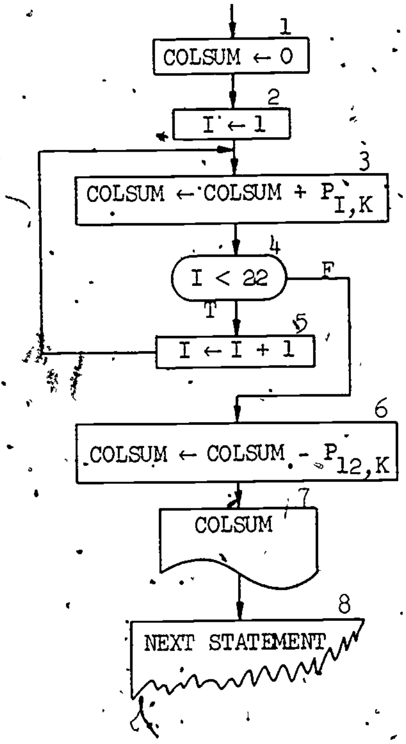
Comment: These exercises involve simple loops operating on elements of a row or a column of an array. Exercises 2 through 5 are fundamental row operations of matrix algebra. Working these will be very helpful in understanding some of the steps in the algorithms described in Section 7-5 (Simultaneous linear equations).

Answers to Exercises 3-6

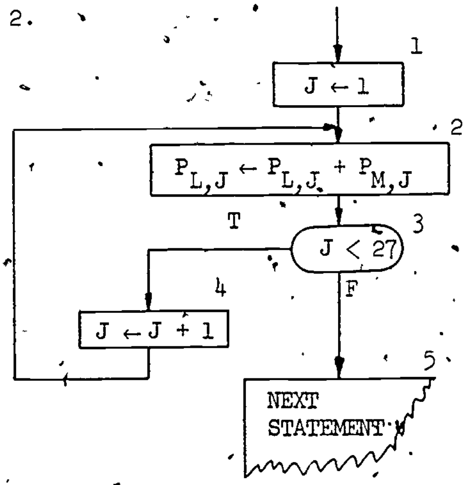
1.



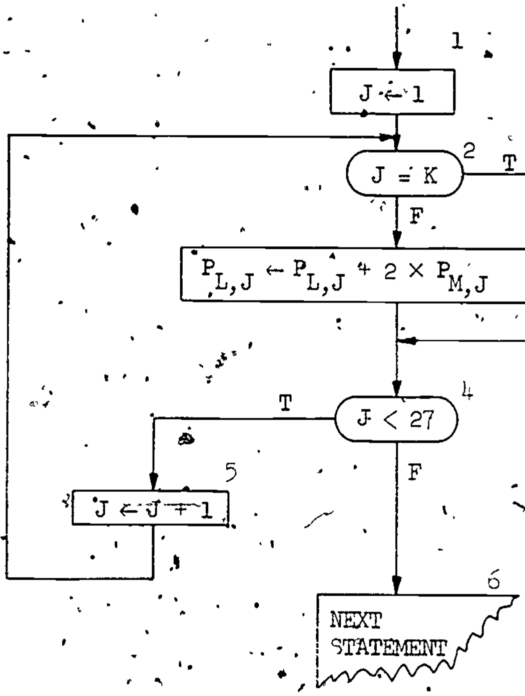
Alternate solution:



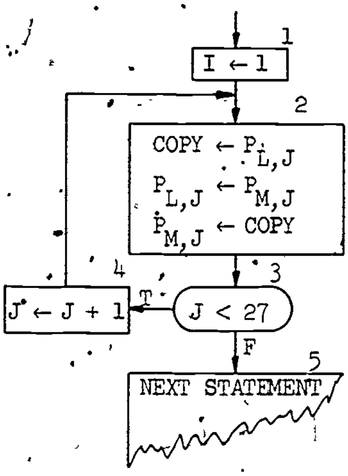
2.



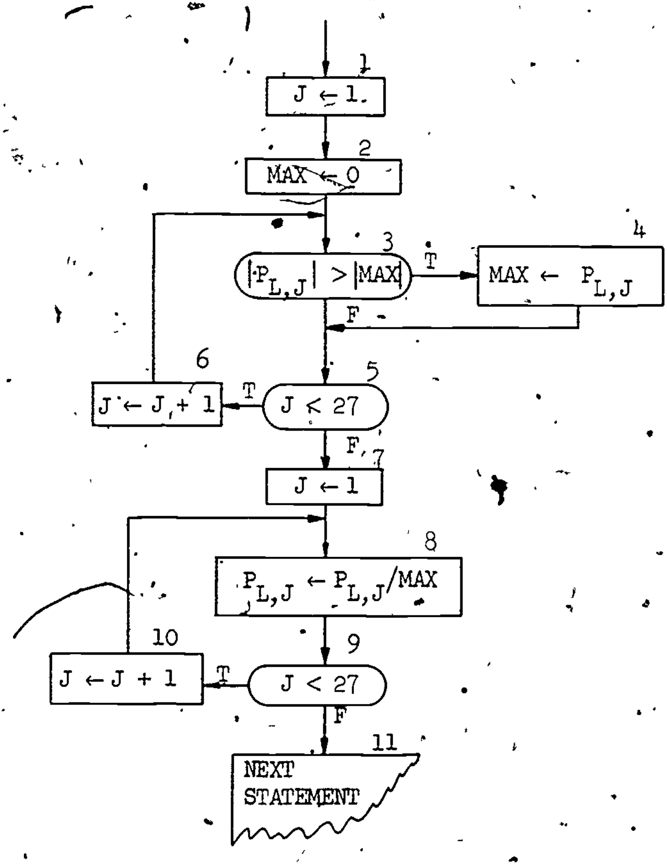
3.


1
$J \leftarrow 1$
2
$J = K$
T
F
3
$P_{L,J} \leftarrow P_{L,J} + 2 \times P_{M,J}$
4
$J < 27$
T
F
5
$J \leftarrow J + 1$
6
NEXT STATEMENT

4.


1
I ← 1
2
COPY ← $P_{L,J}$
$P_{L,J}$ ← $P_{M,J}$
$P_{M,J}$ ← COPY
3
J < 27
T
4
J ← J + 1
F
5
NEXT STATEMENT

5.


1
J ← 1
2
MAX ← 0
3
$|P_{L,J}| > |MAX|$
T
4
MAX ← $P_{L,J}$
F
5
J < 27
T
6
J ← J + 1
F
7
J ← 1
8
$P_{L,J}$ ← $P_{L,J}$/MAX
9
J < 27
T
10
J ← J + 1
F
11
NEXT STATEMENT

Chapter T4

LOOPING

The four sections of this chapter are

4-1 Looping

4-2 Illustrative Examples

4-3 Table-Look-Up

4-4 Nested Loops

Outline of the Chapter

4-1 The ability to describe repetitive events or loops in a convenient unambiguous way is one of the most important skills in programming. Here students develop some insight into what loops are. Although there is no single form which all loops take, there is one kind of loop so common as to merit the development of shorthand techniques for describing it. This is the main reason for the "iteration box" introduced in this section and employed numerous times thereafter in this text.

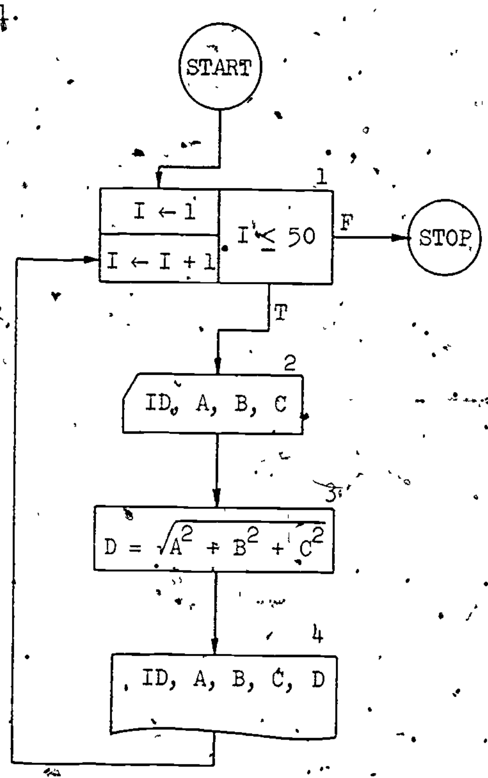
4-2 This section includes many small problems each of which involves a loop best described using an iteration box.

4-3 Here we take up the simple topic of table-look-up and treat it in depth. We look at many aspects of this one important problem and bring a number of programming concepts into focus.

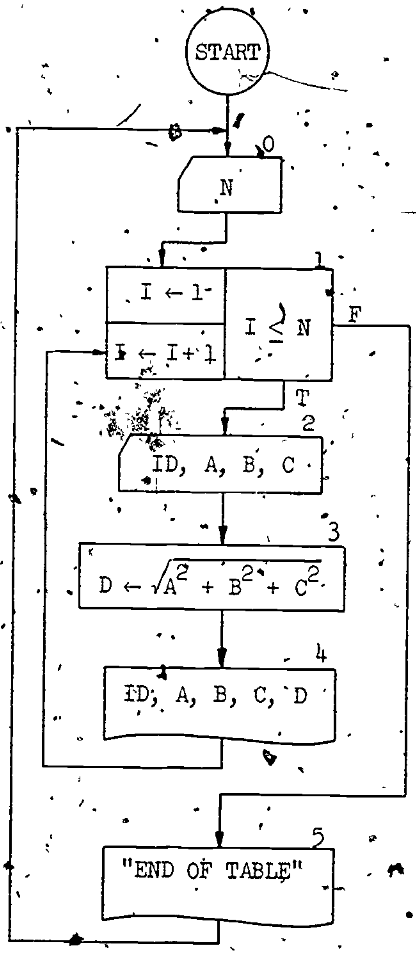
4-4 We introduce the nesting of loops or repetitions. This is an all important idea. Repetitions frequently come not singly, but in bunches and often one within another. Many operations on matrices fall in this category. Sorting does, too. One of the most interesting algorithms in the entire text, that of finding a longest monotone subsequence, is described at the end of this section. The algorithm suggests the proof of an interesting theorem about the minimum length of a monotone subsequence. Without first focusing on the algorithm, it is hard to see how the proof would ever suggest itself.

Answers to Exercises 4-1

1.



2.



1. Check values. Let I ≤ 5.

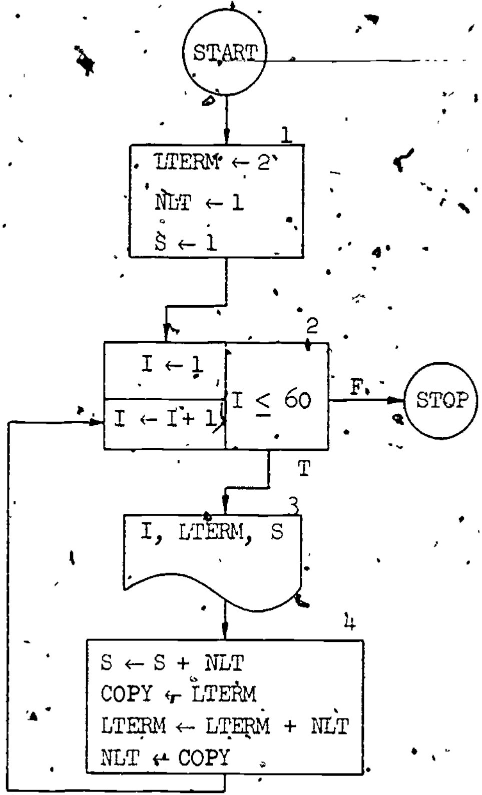
Input

ID	A	B	C
1	3	4	5
2	6	8	10
3	9	12	15
4	12	16	20
5	15	20	25

Output

```
1,   3,   4,   5,    7.07
2,   6,   8,  10,   14.14
3,   9,  12,  15,   21.21
4,  12,  16,  20,   28.28
5,  15,  20,  25,   35.35
```

3.



Comment:

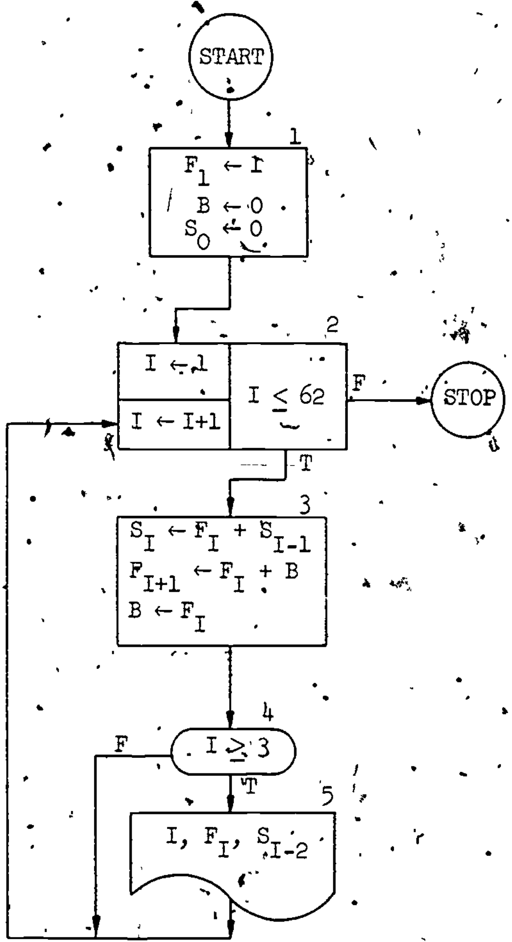
The computed results are:

I	LTERM	S
1	2	1
2	3	2
3	5	4
4	8	7
5	13	12
6	21	20
	etc.	

The check value $I \leq 5$

output 5, 13, 12

Alternate solution using subscripts:

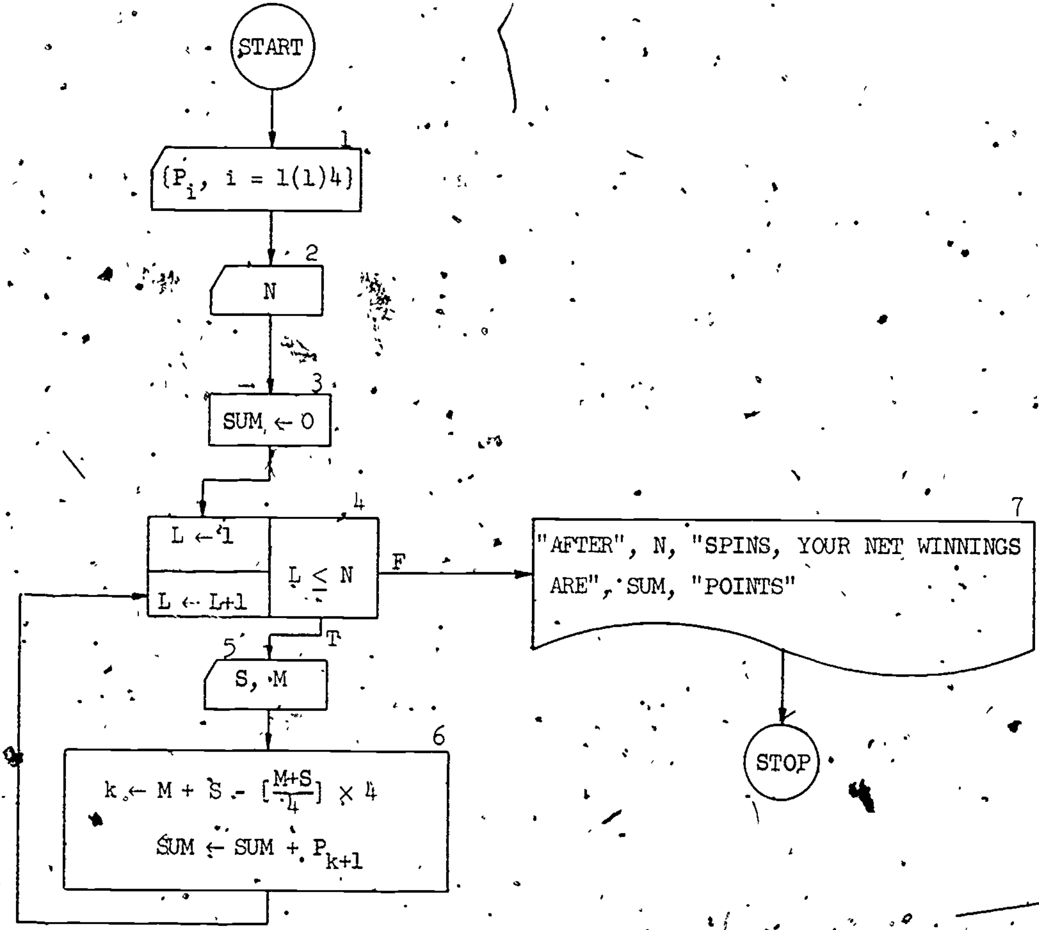


Comment: Computed results are:

I	F_I	S_{I-2}
3	2	1
4	3	2
5	5	4
6	8	7
7	13	12
8	21	20
	etc.	

Notice that LTERM and S differ by 1 for all I ! Or, in the alternate solution, F_I and S_{I-2} differ by 1 for all I. We hope the student will find this to be an interesting discovery. Also notice subscripts are not essential in this flow chart since the first solution does not use them.

4.


START
1
$\{P_i, i = 1(1)4\}$
2
N
3
SUM ← 0
4
L ← 1
L ← L+1
L ≤ N
F
T
5
S, M
6
$k \leftarrow M + S - [\frac{M+S}{4}] \times 4$
$SUM \leftarrow SUM + P_{k+1}$
7
"AFTER", N, "SPINS, YOUR NET WINNINGS ARE", SUM, "POINTS"
STOP

Check values: N = 5

Input	S,	M
1,	21,	216
2,	30,	126
3,	24,	621
4,	15,	261
5,	11;	612

Note on the carnival wheel:

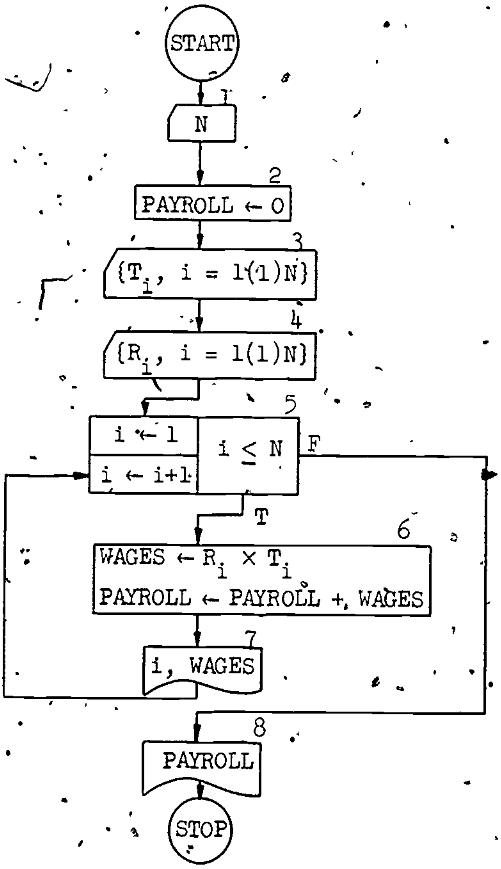
Although we have not done so in this text, one can proceed to simulate some interesting experiments for discovering game-playing characteristics of the wheel. In this exercise we have suggested that the spins in the sequence be supplied as data. It would be more interesting if the computer could operate such data in some random fashion by employing a procedure for generating random numbers. Such a procedure would "generate" a value of m each time it is called on. Values of m might be uniformly distributed over a given range or, more realistically, distributed normally over the same range about some mean value of m which you or your students could vary. Some useful references which will suggest algorithms for generating random numbers are:

1. Problems for Computer Solution by F. Gruenberger and G. Jaffrey, John Wiley, 1965. See Problems E7, E11 and E14.

2. A Fortran IV Primer by E. Organick, Addison-Wesley, 1966. See Problem 5 (Simulation of Experiments).

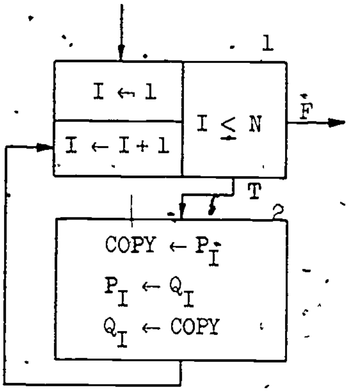
3. The Language of Computers by G. Galler, McGraw-Hill, 1962. See Section 6.3, "Random Number Generators", p. 72.

5.

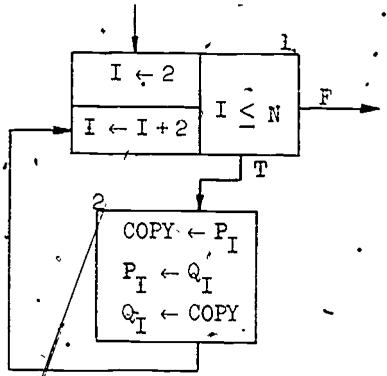


Answers to Exercises 4-2 Set A

1.

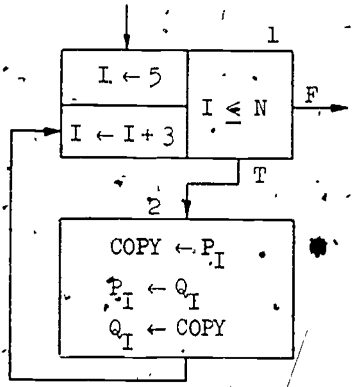


2.

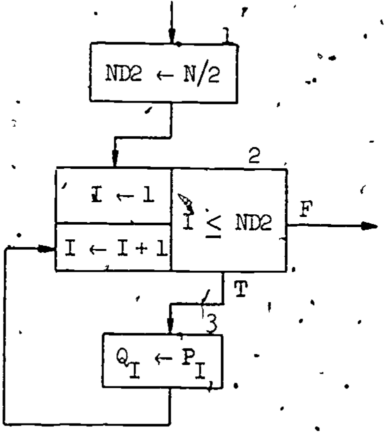


Immaterial whether N is odd or even.

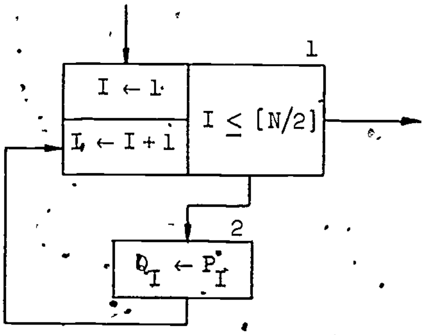
3.



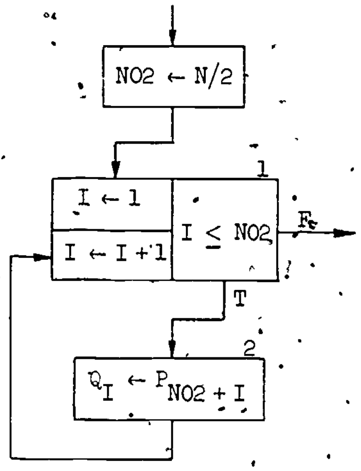
4.



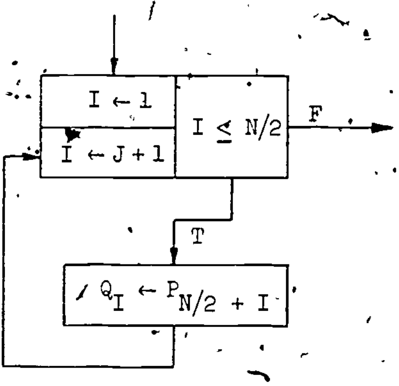
An alternative, though less efficient, way of doing it:



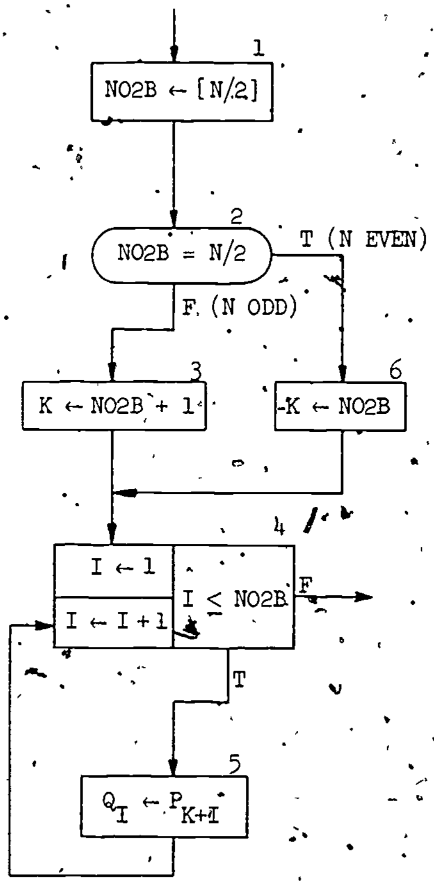
5.



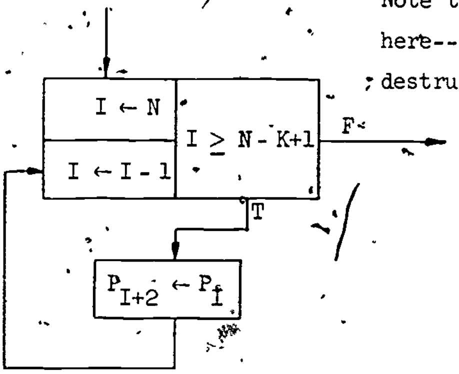
An alternative, though usually less efficient, way of doing it:



6. Perhaps the "cleanest" approach, logically speaking, is to first determine if N is even or odd and then act accordingly.

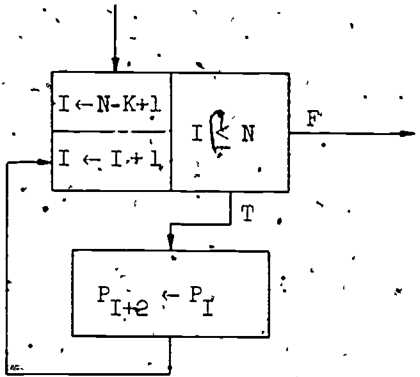


7.

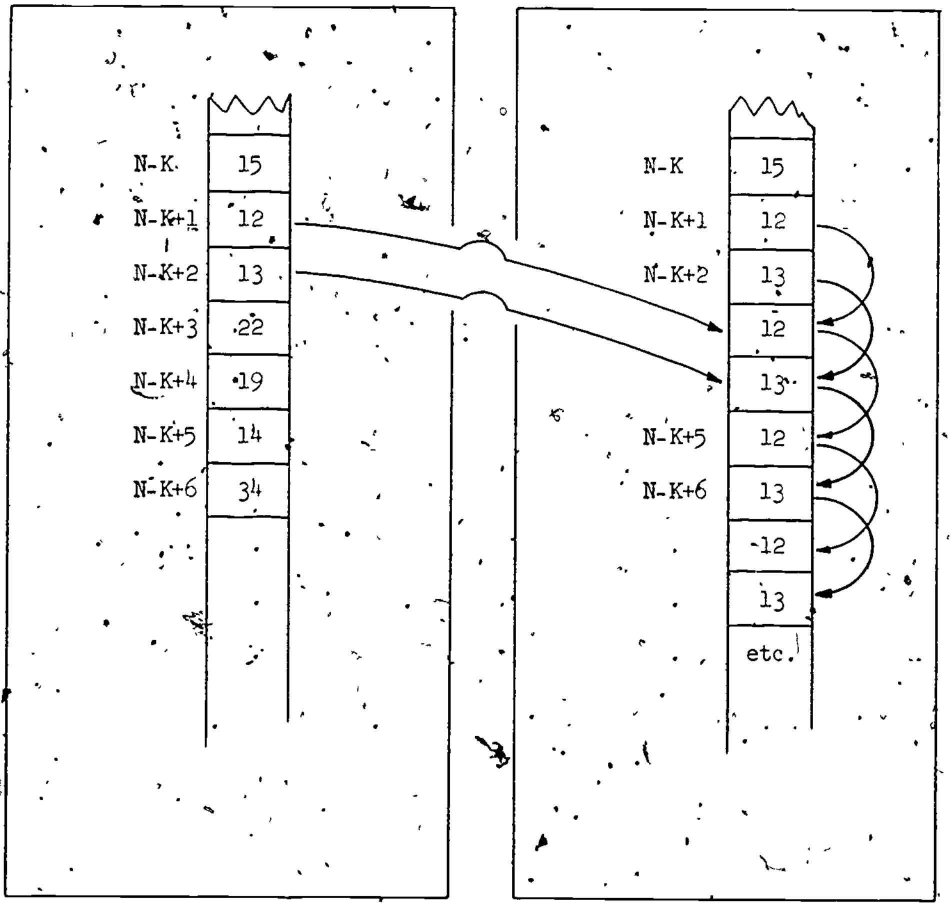


Note that I must be incremented negatively here--to do it any other way would be self destructive.

For example, suppose this flow chart is used:

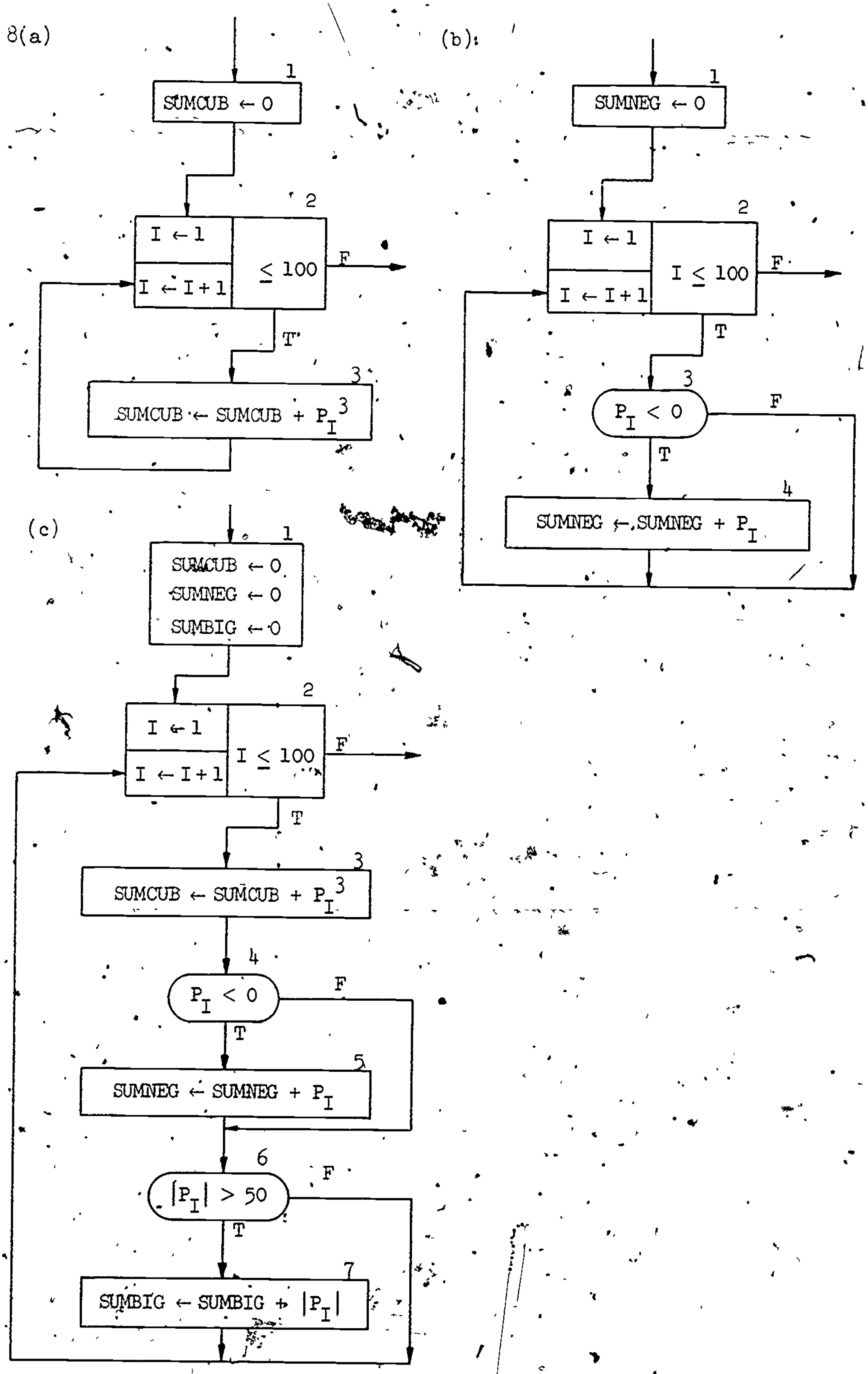


By virtue of destructive read in, the pair of values originally assigned to P_{N-K+1} and P_{N-K+2} would be repeated over and over again. See Fig. 4 - 13.



Before the move

After the disaster



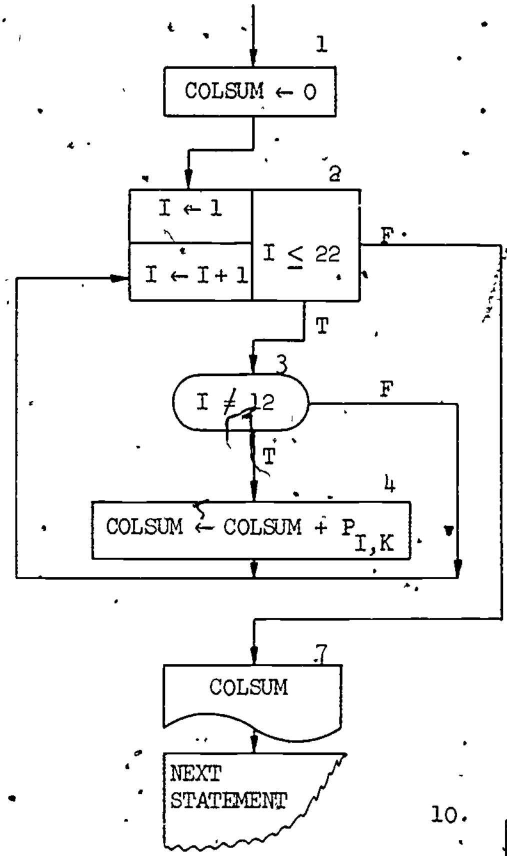
8(a)

(b)

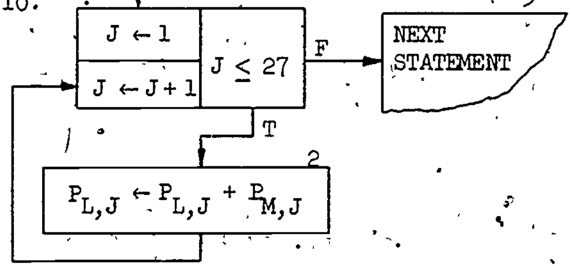
(c)

1
SUMCUB ← 0
2
I ← 1
I ← I+1
≤ 100
F
T
3
SUMCUB ← SUMCUB + P_I^3
1
SUMNEG ← 0
2
I ← 1
I ← I+1
I ≤ 100
F
T
3
$P_I < 0$
F
T
4
SUMNEG ← SUMNEG + P_I
1
SUMCUB ← 0
SUMNEG ← 0
SUMBIG ← 0
2
I ← 1
I ← I+1
I ≤ 100
F
T
3
SUMCUB ← SUMCUB + P_I^3
4
$P_I < 0$
F
T
5
SUMNEG ← SUMNEG + P_I
6
$|P_I| > 50$
F
T
7
SUMBIG ← SUMBIG + $|P_I|$

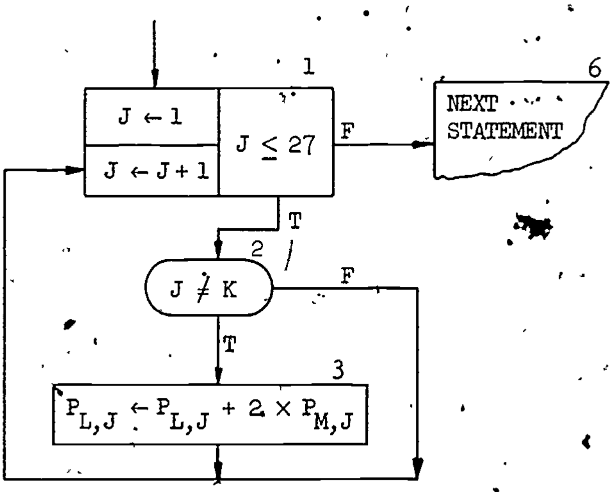
9.


1
COLSUM ← 0
2
I ← 1
I ← I + 1
I ≤ 22
F
T
3
I ≠ 12
F
T
4
COLSUM ← COLSUM + P_{I,K}
7
COLSUM
NEXT STATEMENT

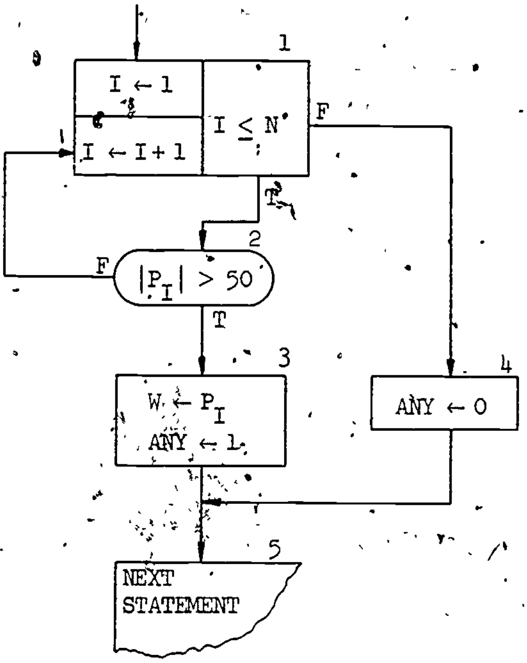
10.


1
J ← 1
J ← J + 1
J ≤ 27
F
5
NEXT STATEMENT
T
2
P_{L,J} ← P_{L,J} + P_{M,J}

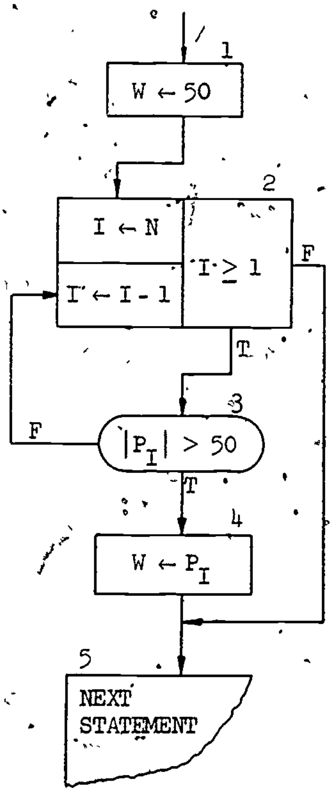
11.


1
J ← 1
J ← J + 1
J ≤ 27
F
6
NEXT STATEMENT
T
2
J ≠ K
F
T
3
P_{L,J} ← P_{L,J} + 2 × P_{M,J}

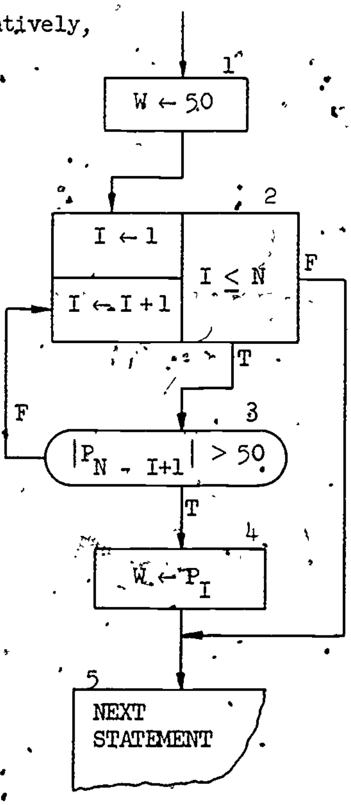
12.



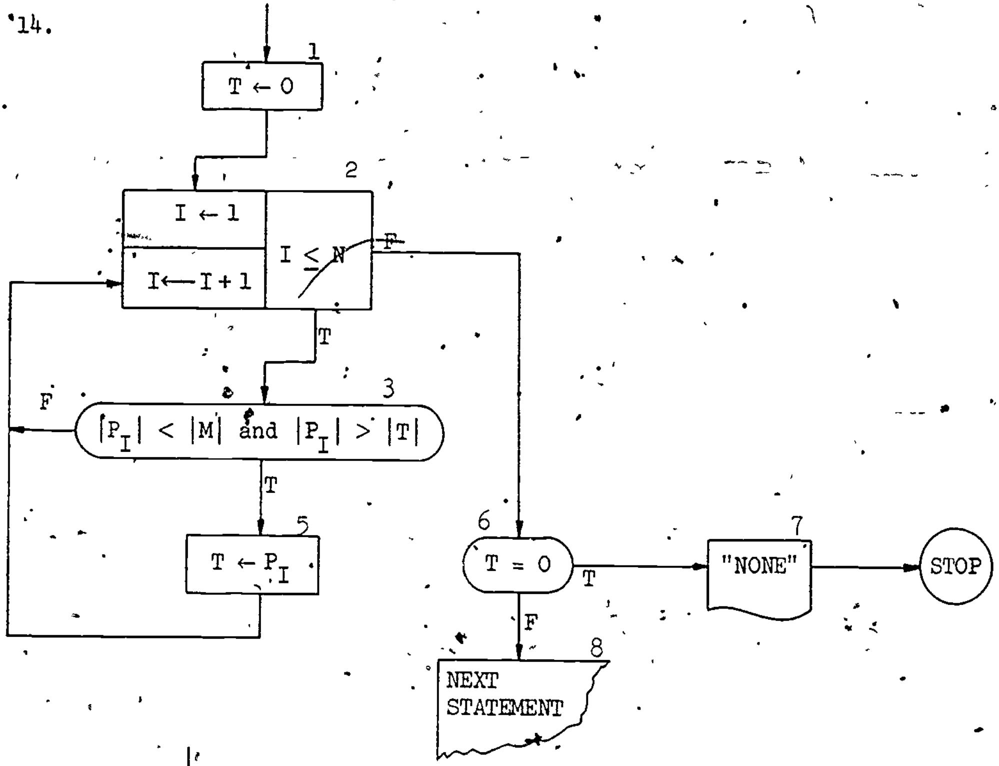
13.



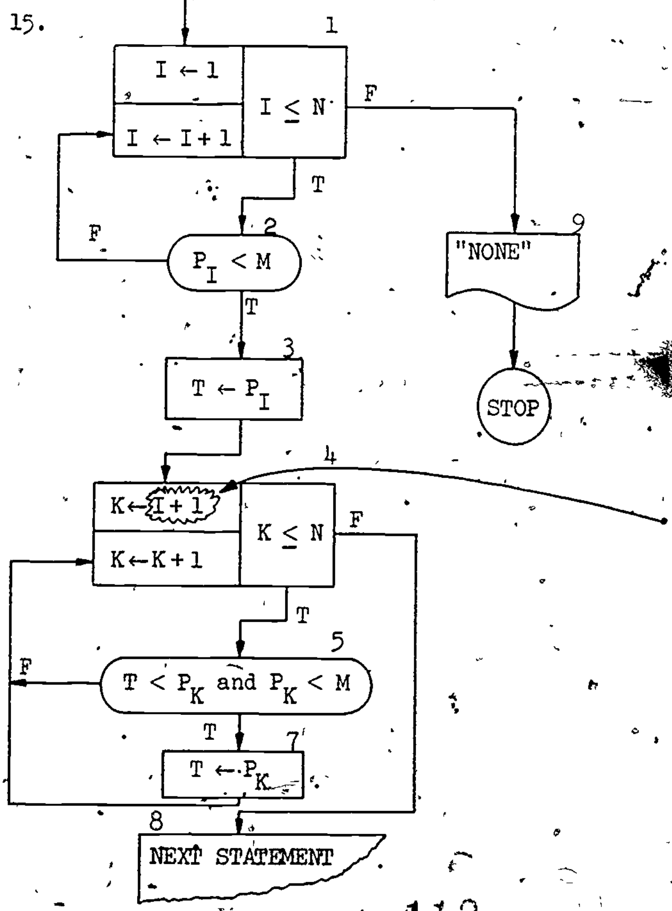
alternatively,



14.

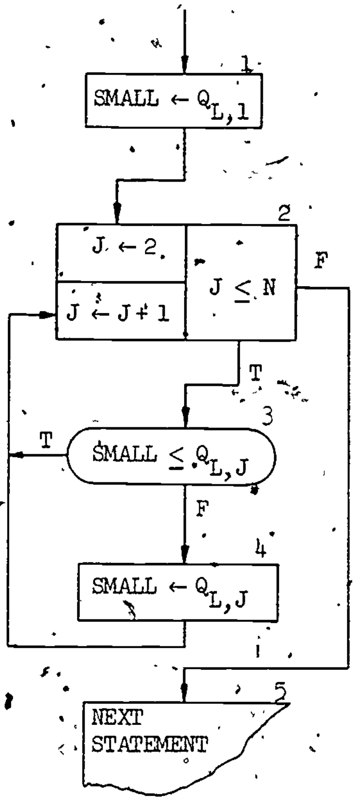


15.

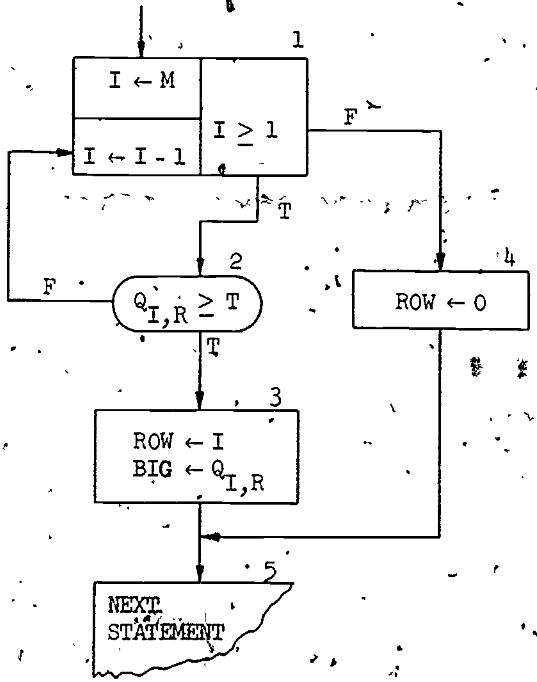


note: we resume the scan down the list

16.

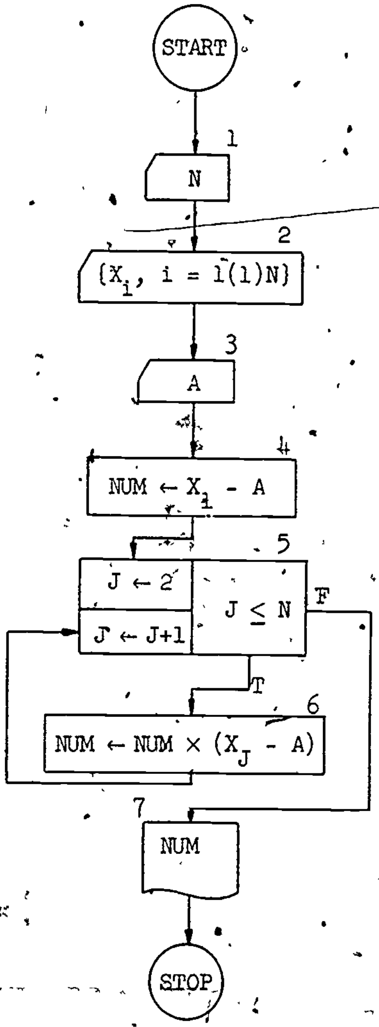

1
SMALL ← $Q_{L,1}$
2
J ← 2
J ← J + 1
J ≤ N
F
T
3
T
SMALL ≤ $Q_{L,J}$
F
4
SMALL ← $Q_{L,J}$
5
NEXT STATEMENT

17.


1
I ← M
I ← I - 1
I ≥ 1
F
T
2
F
$Q_{I,R}$ ≥ T
T
4
ROW ← 0
3
ROW ← I
BIG ← $Q_{I,R}$
5
NEXT STATEMENT

Answers to Exercises 4-2 Set B

1(a)



Check values.

Input

$N = 5$

$X_1 = 3$

$X_2 = 18$

$X_3 = -7$

$X_4 = 62$

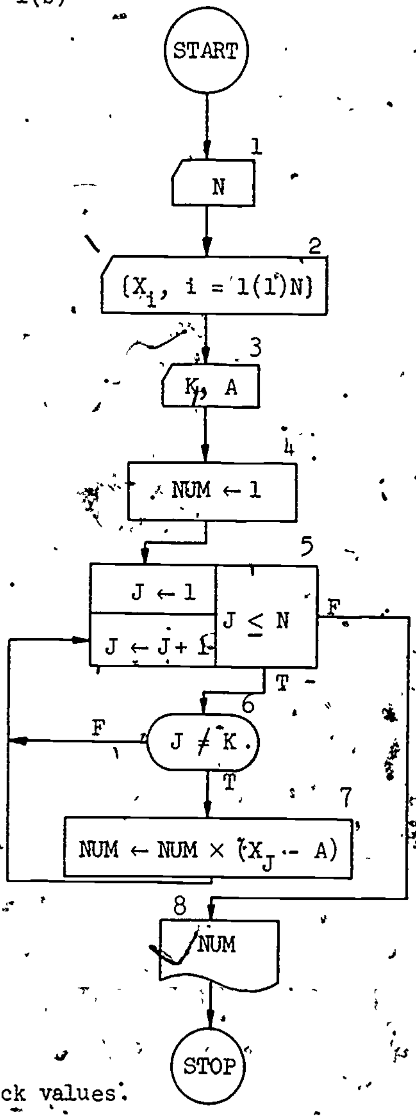
$X_5 = -19$

$A = 22$

Output

NUM = 3,614,560

1(b)



Check values.

Input

$X_1 = 3$

$X_2 = 18$

$X_3 = -7$

$X_4 = 62$

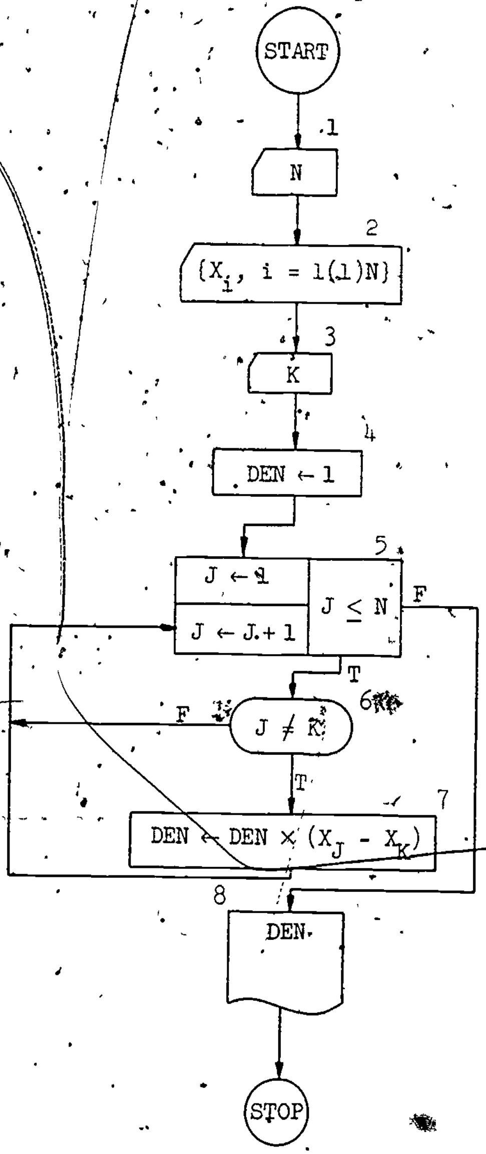
$X_5 = -19$

$A = 22$ $K = 4$

Output

NUM = 90, 364

2. The value of A is no longer needed as input data. X_K may be used in its place (Box 6). Otherwise, the computation of NUM in 1(b), and that of DEN, is identical.



Check values

Input

X_1 = 3

X_2 = 18

X_3 = -7

X_4 = 62

X_5 = -19

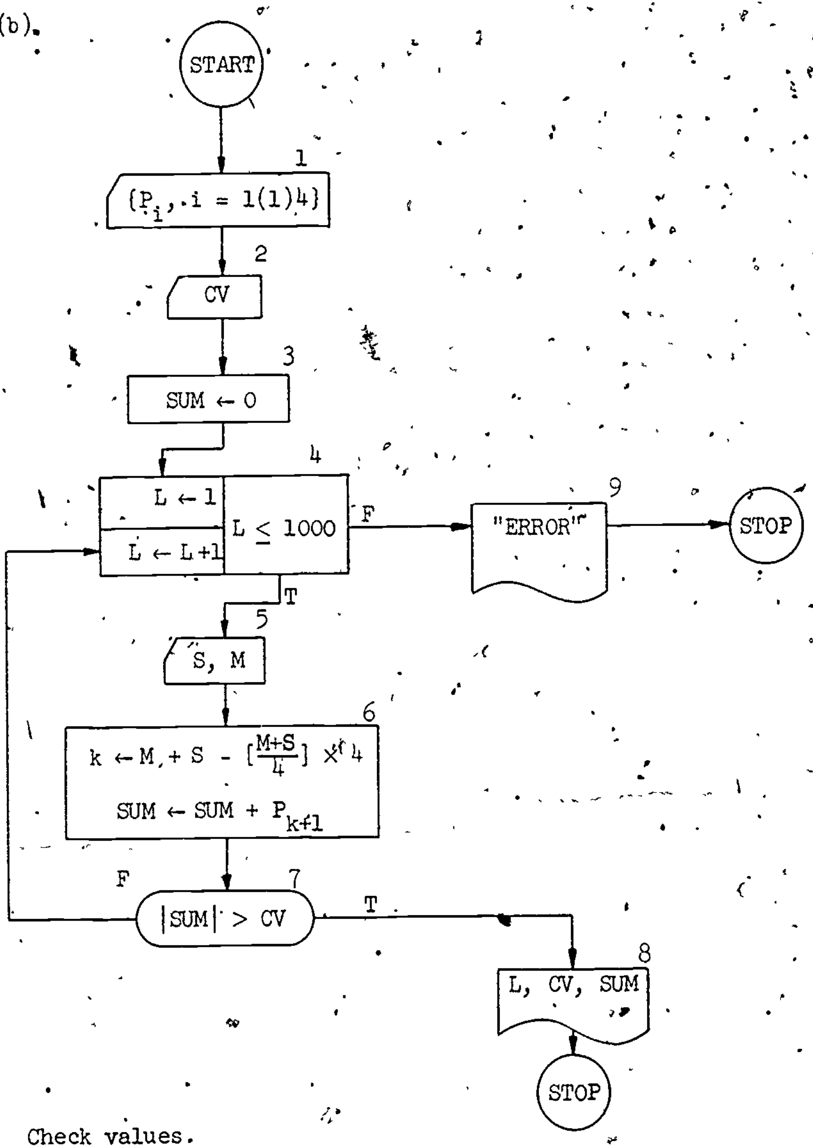
K = 2

Output

NUM = -610,500

3(a) Same as answer to Exercise 3 Section 4-1.

(b)



Check values.

Input Use data cards from Ex. 4, 4-1

N = 5

M	S		
216,	21	CV = 5	Output 4, 5, 6
126,	30		
621,	24		
261,	15		
612,	11		

Answers to Exercises 4-3 Set A

The same 2-point formula for a straight line is

$$y - y1 = \frac{y2 - y1}{x2 - x1} \times (x - x1).$$

Applying this, we have

$$YINT \doteq \left(\frac{Y_I - Y_{I-1}}{X_I - X_{I-1}}\right) \times (A - X_{I-1}) + Y_{I-1}$$

where we have let $y = YINT$

$y1 = Y_{I-1}$

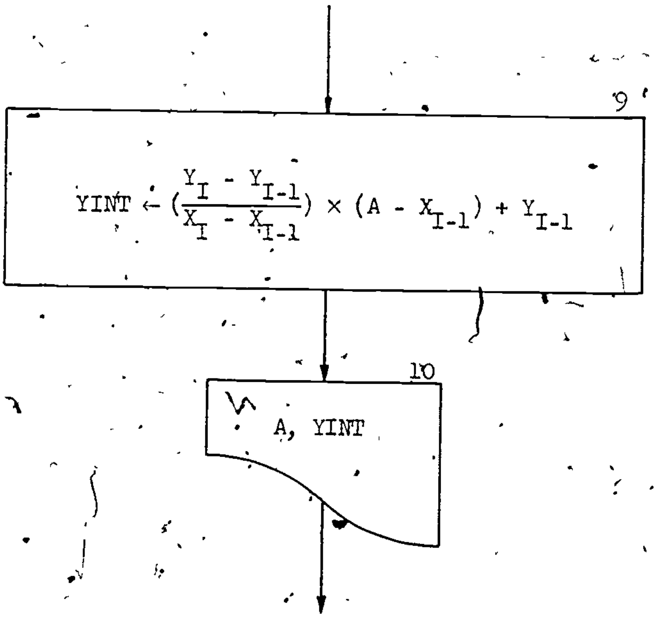
$y2 = Y_I$

$x1 = X_{I-1}$

$x2 = X_I$

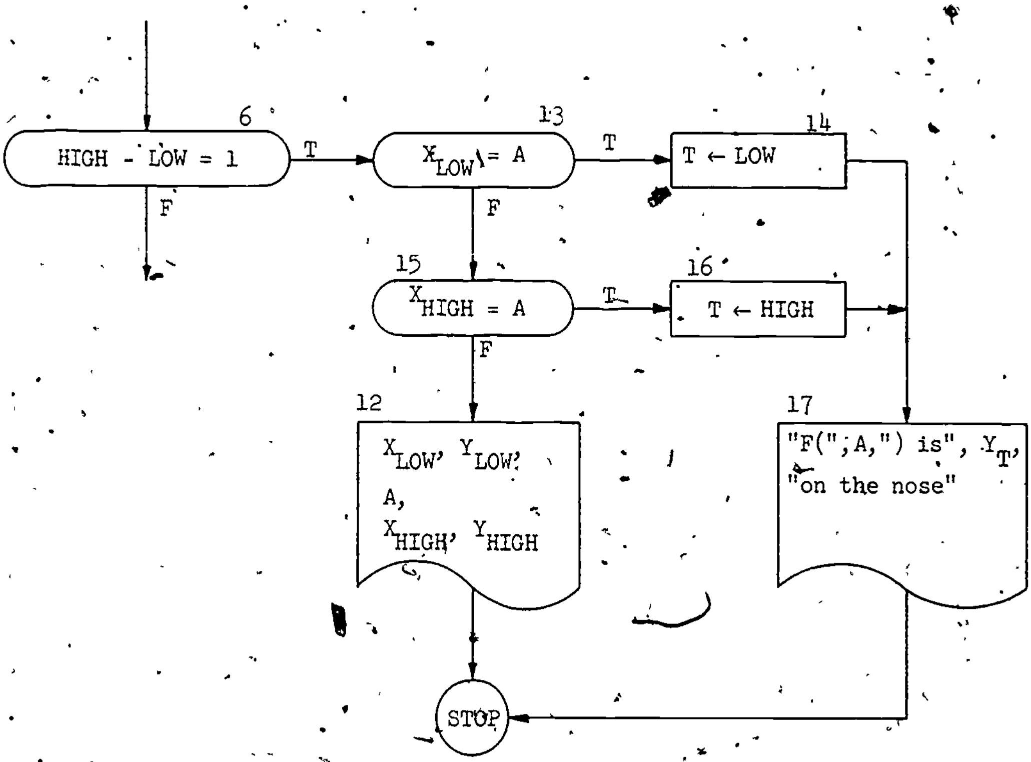
and $x = A$.

We may then replace Box 9 of Figure 4-22 with


9
YINT ← $\left(\frac{Y_I - Y_{I-1}}{X_I - X_{I-1}}\right) \times (A - X_{I-1}) + Y_{I-1}$
10
A, YINT

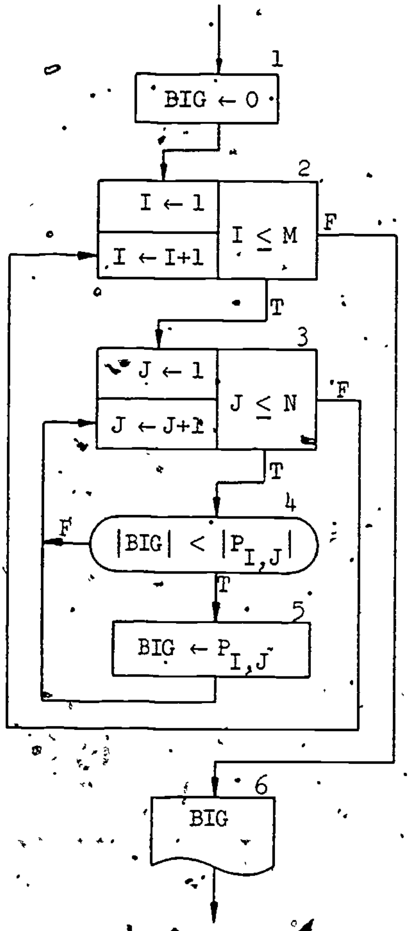
Answer to Exercise 4-3 Set B

The new Boxes 13 through 17 are to be added as shown:

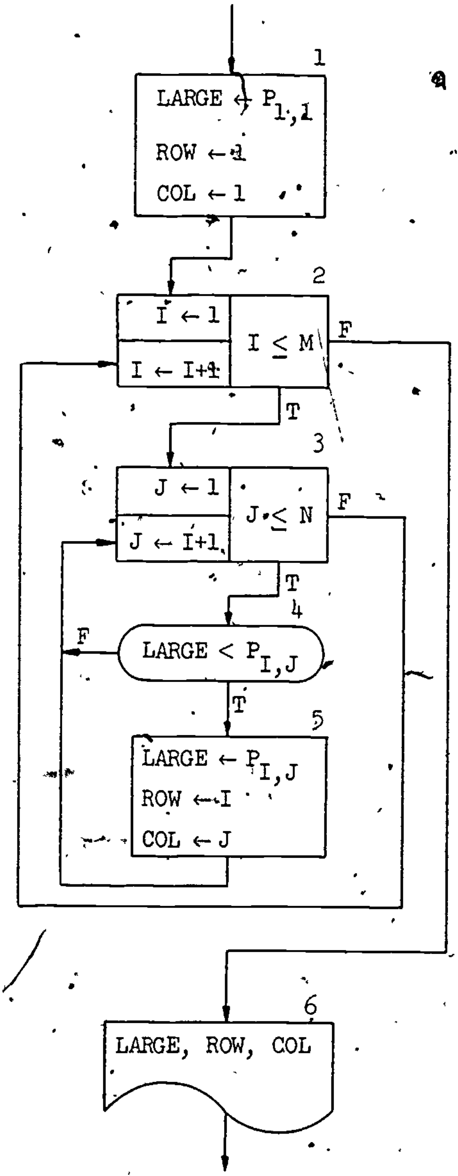


Answers to Exercises 4-4 Set A

1.



2.



Check values.

Input

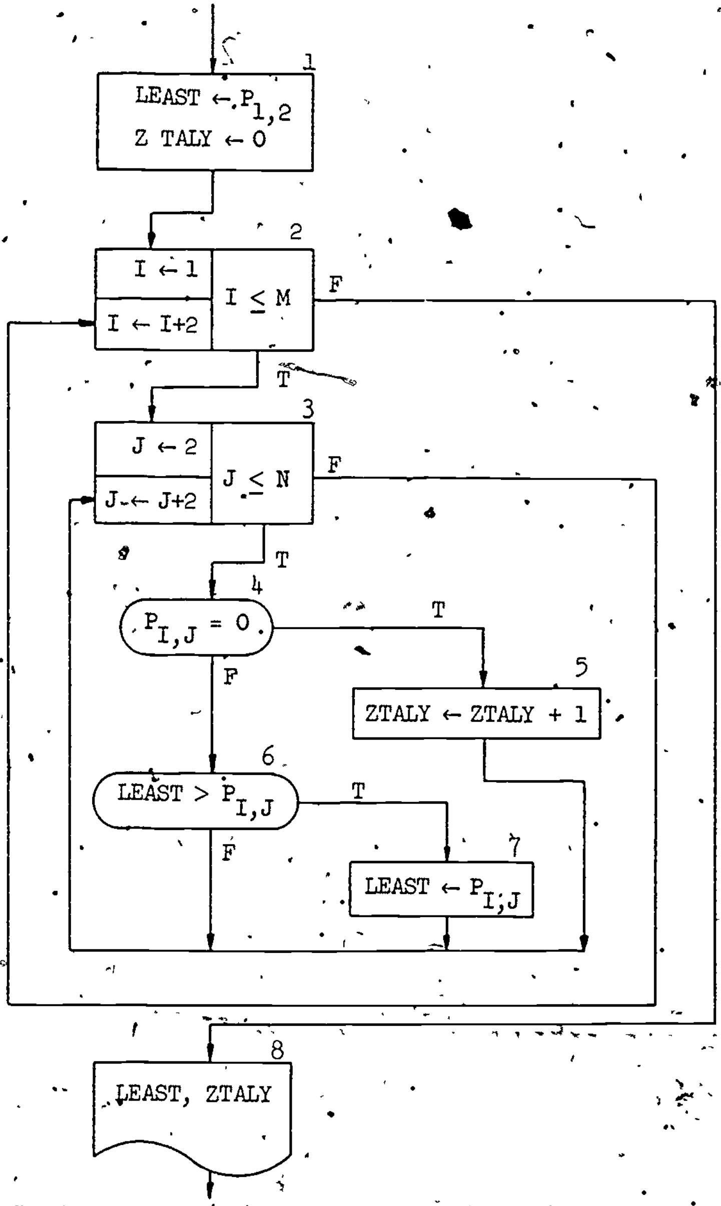
Card 1	2	3	0
Card 2	0	7	9
Card 3	12	10	19
Card 4	18	16	12

Output

LARGE, ROW, COL

19, 3, 3

3.



Check values. (Use cards from Ex. 4-4, 2)

Input

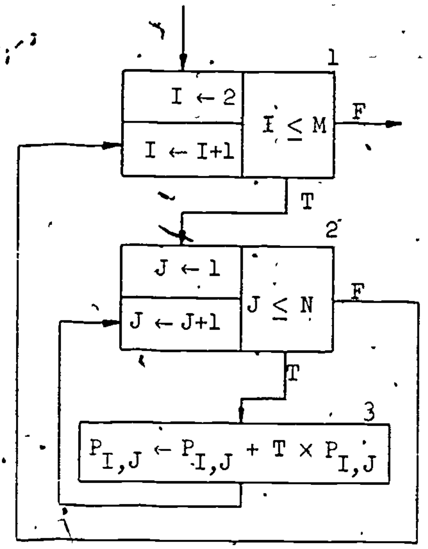
Card 1	2	3	0
Card 2	0	7	9
Card 3	12	10	19
Card 4	18	16	12

Output

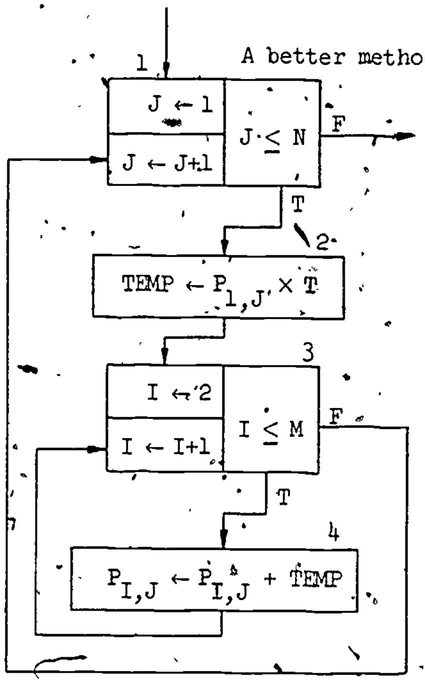
LEAST, ZTALY

3, 0

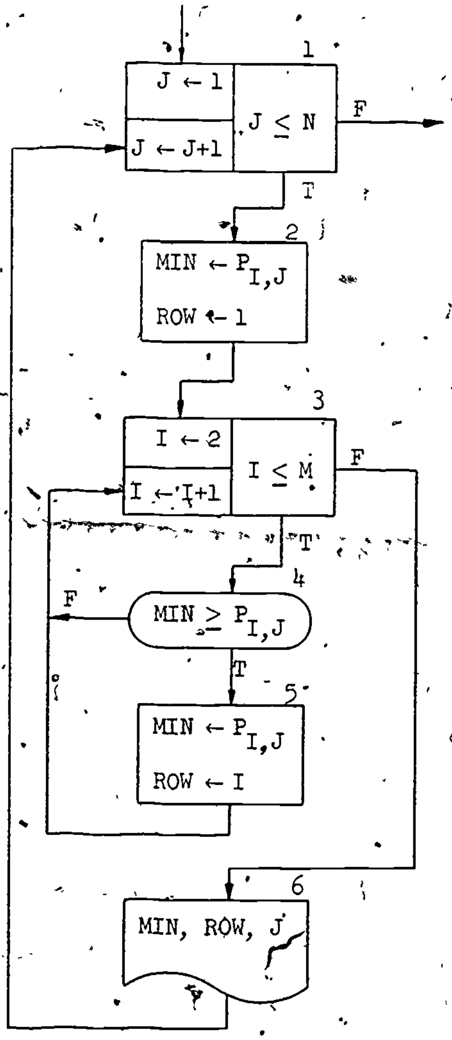
4.



A better method



5.



Check values.

(Use cards from Ex. 4-4, 2).

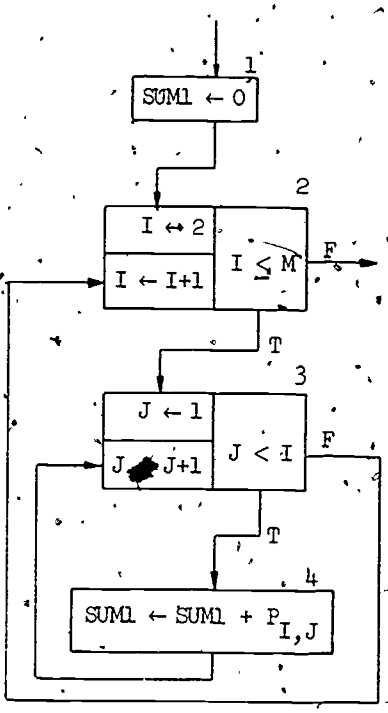
Input

Card 1	2	3	0
Card 2	0	7	9
Card 3	12	10	19
Card 4	18	16	12

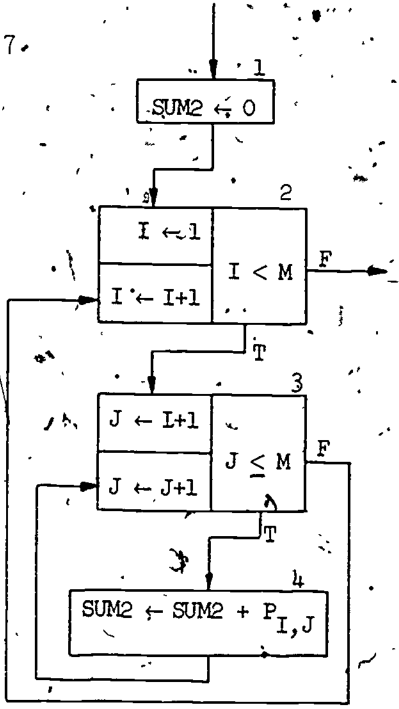
Output

MIN	ROW	COL
0	2	1
3	1	2
0	1	3

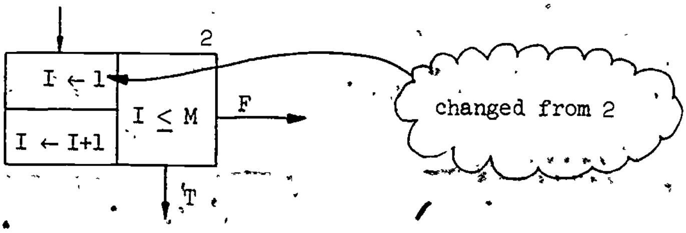
6.



7.

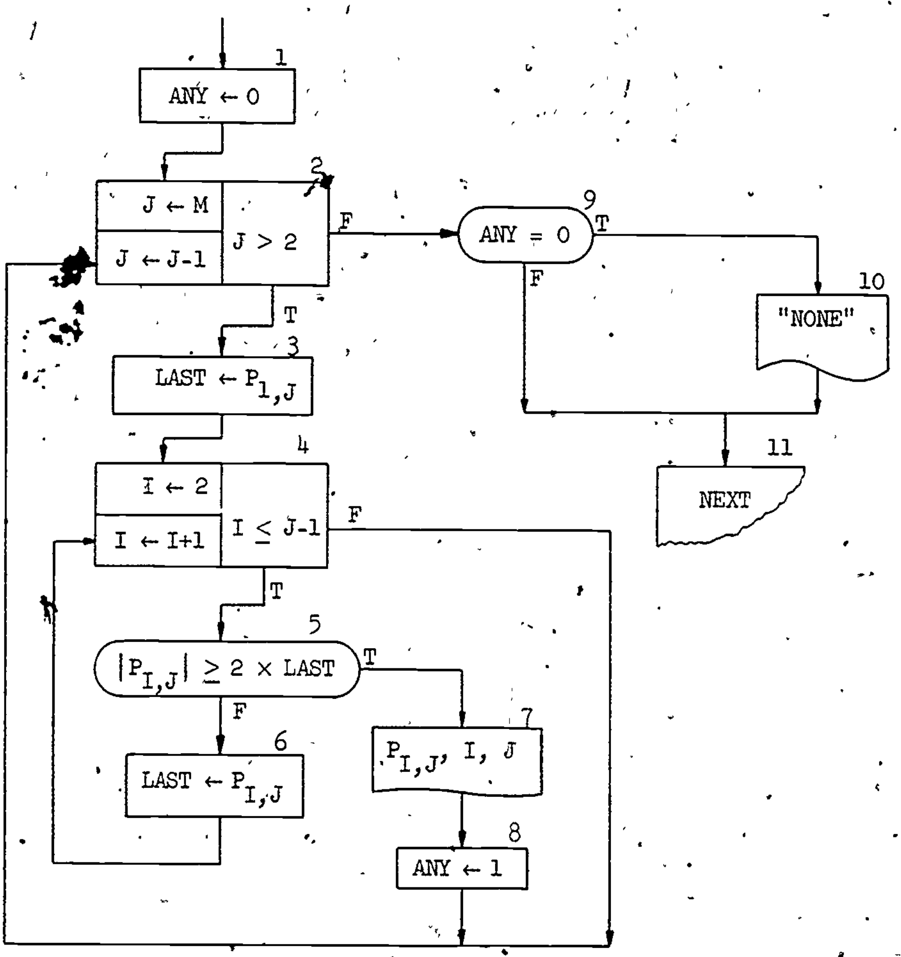


Comment on No. 6: If the student draws Box 2 as



it is not wrong. No entry for row 1 will be taken anyway. When Box 3 is executed for the first time, J is set to 1 and then the test, J < I, is made. Of course, it will be false because both I and J are 1. We then exit from the inner loop immediately with no execution of Box 4.

8.



Searching for Pigs

Check values. (Use last 3 cards from Ex. 4-4, 2)

Input

Card 2	0	7	9
Card 3	12	10	19
Card 4	18	16	12

M = 3

Output	19	2	3

Input Check values 6 × 6

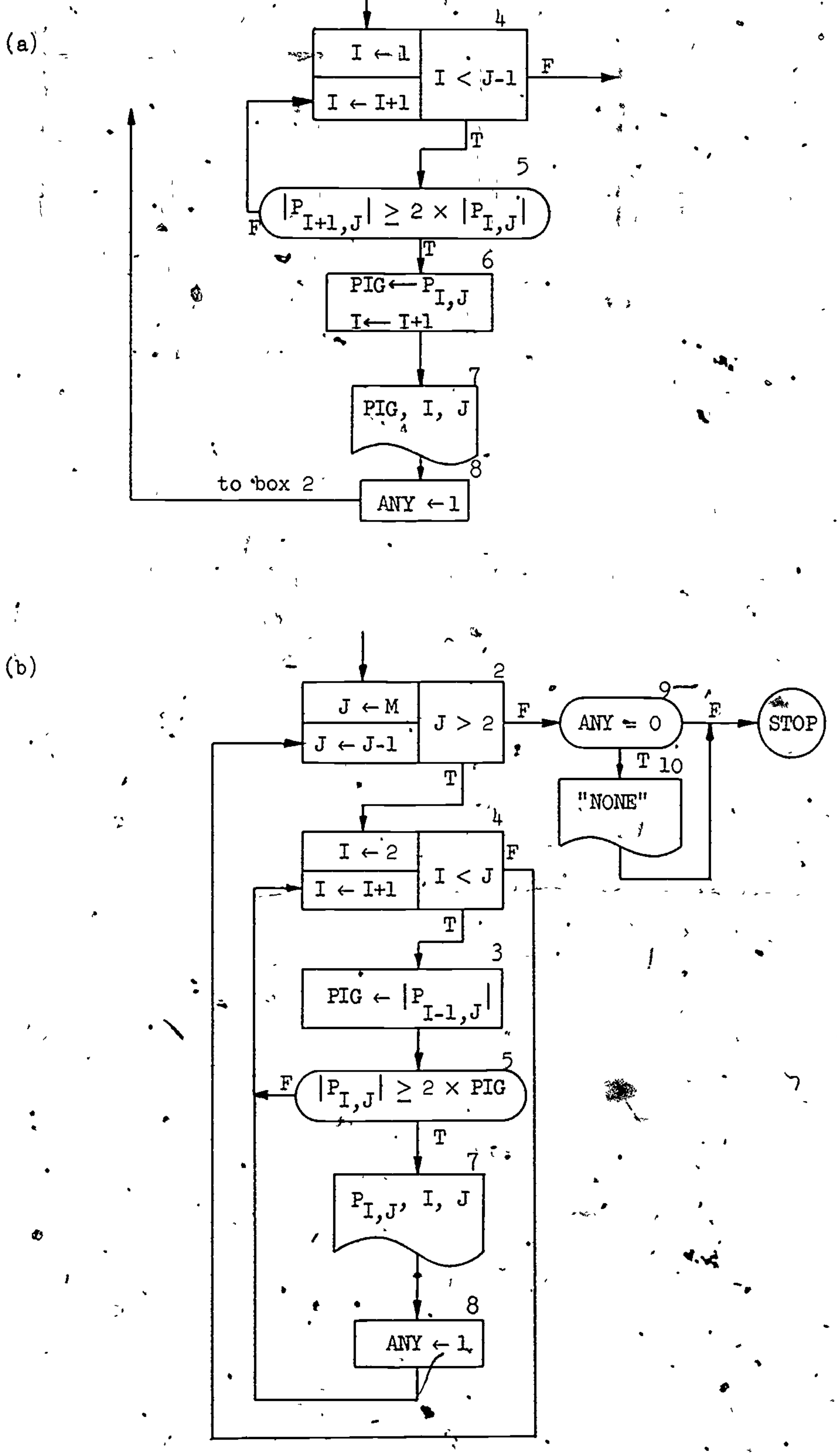
M = 6

Card 1	2	1	0	3	4	2
Card 2	2	3	2	7	3	6
Card 3	7	2	-3	8	9	4
Card 4	8	1	4	2	-3	8
Card 5	6	.5	2	1	1	1
Card 6	4	-2	1	1	2	1

Output

$P_{I,J}$	I	J
6	2,	6
8	4,	6
9	3,	5
7	2,	4
2	2,	3

. Alternative approaches to solution of Problem 8.



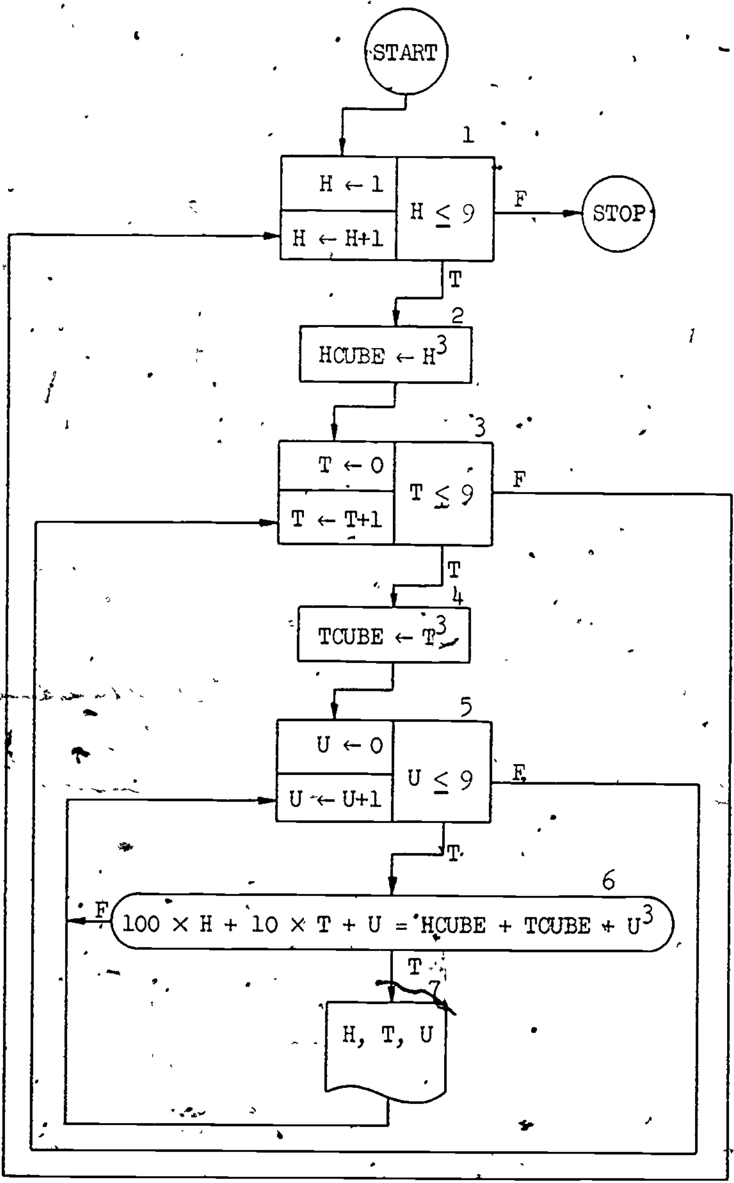
Answers to Exercises 4-4 Set B

1. 8 multiplications each time Box 4 is executed.
   Box 4 is executed 900 times, so the answer is 8×900 or 7200 times.

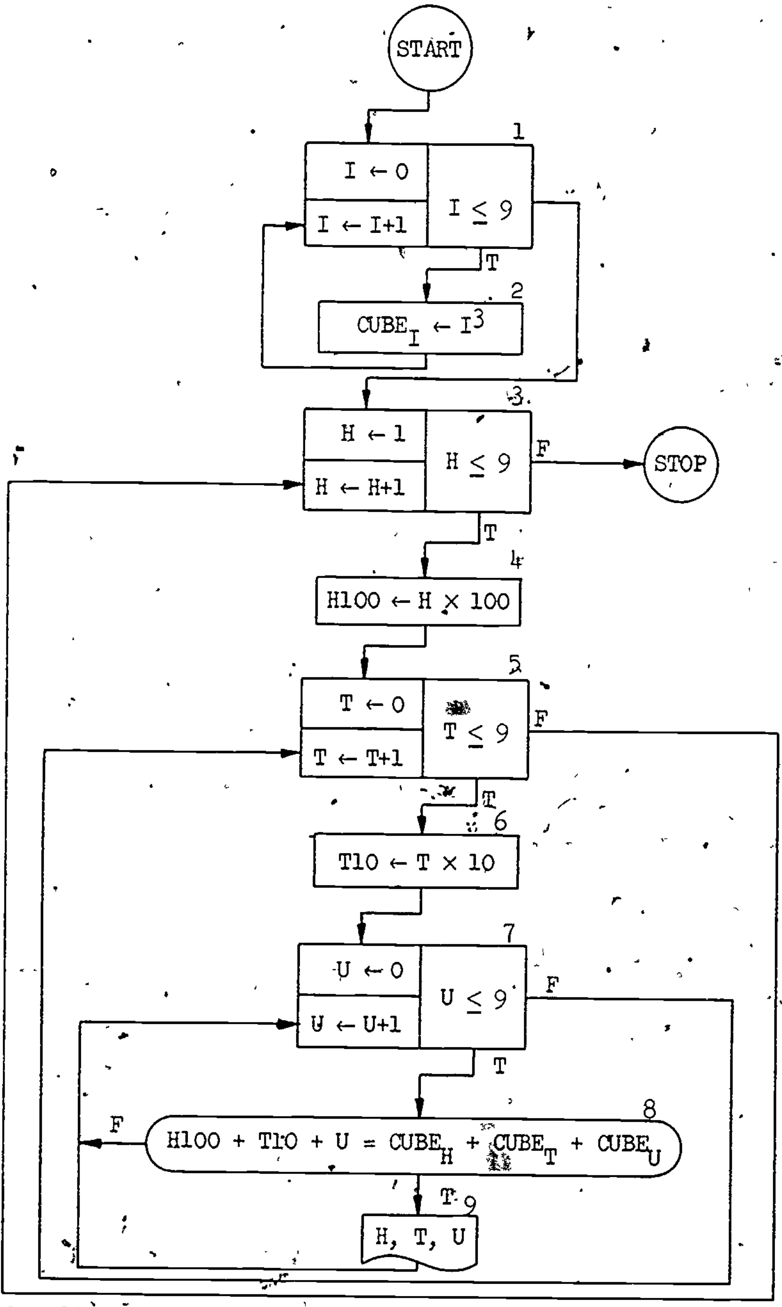
2. Only 9 different values of H^3.

3. Only 10 different values of T^3.

4.



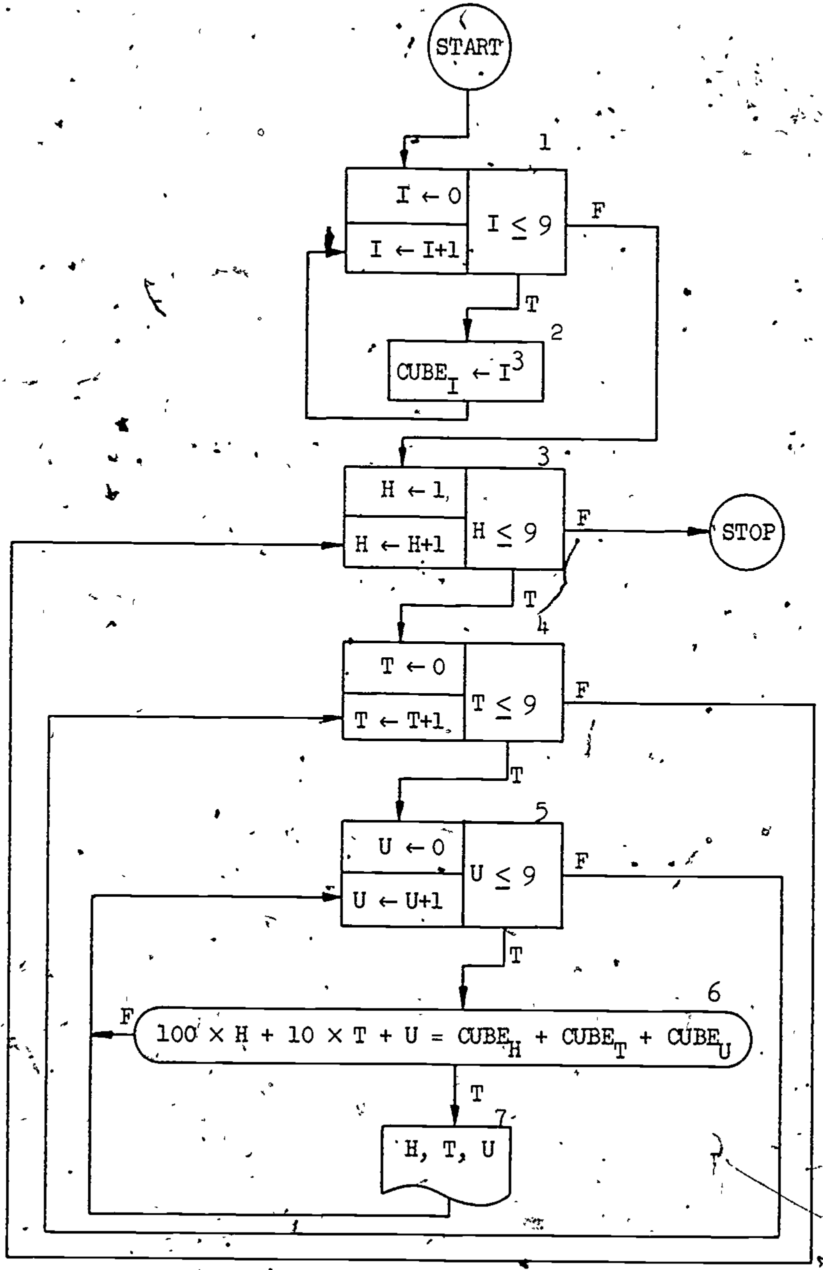
6. One could even further reduce the number of multiplications as shown on the next flow chart. It would be interesting to see how many students suggest this further improvement on their own. Note that in making these changes we add boxes to our flow chart which in computer programming means more instructions in the program.



Multiplications:	20 (Box 2)	+	9 (Box 4)	+	90 (Box 6)	= 119

versus 7200 in the original algorithm.

5.



Multiplications: 20 (Box2) + 2 × 900 (Box 6) = 1820 versus 7200 in the original flow chart.

7. Solution. The main mathematical point here is the triangle inequality that the length of one side of a triangle is less than the sum of the lengths of the other two. Let I be the length of the longest side of a triangle, J the length of the second longest, and K the length of the shortest, so that

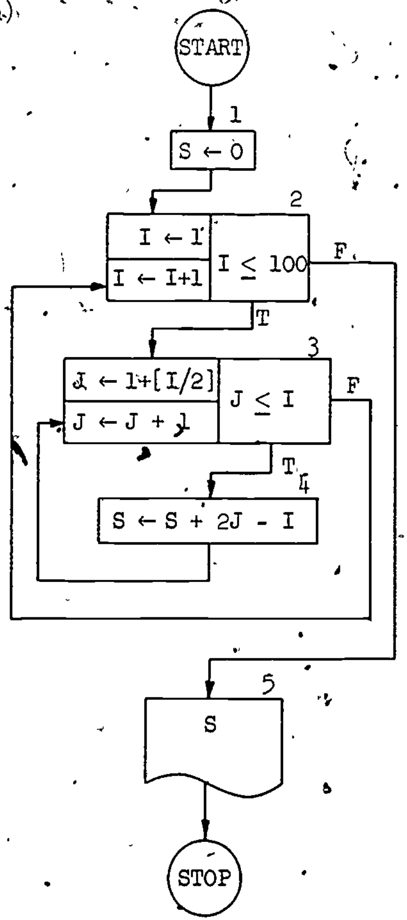
$$1 \leq K \leq J \leq I \leq 100.$$

The triangle inequality yields

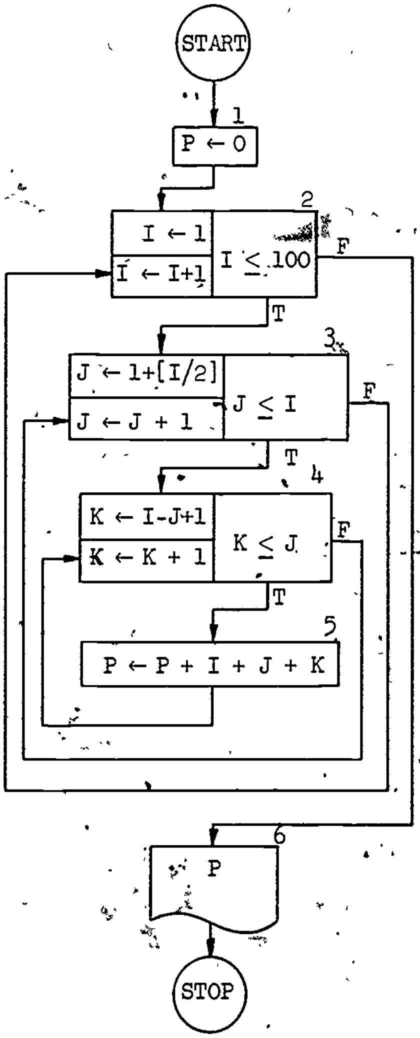
$$J > I/2 \quad \text{and} \quad K > I - J.$$

We wish to count all triples, I, J, K, subject to the above conditions. Once I and J are chosen, K may run from I - J + 1 to J, inclusiv. There are thus J - (I - J + 1) + 1 or 2J - I triangles having the giv values of I and J. We sum this value first for J between [I/2] + 1 to I and then for I between 1 and 100.

(a)



(b)



7. (c) In the solution of part (b) replace P by S in Box 1, replace the condition in Box 2 by "$I < 50$", the condition in Box 4 by "$K \leq J$ and $I + J + K \leq 100$". Replace the assignment in Box 5 by "$S \leftarrow S+1$". Replace P in Box 6 by S .

(d) In the solution to part (b), replace the conditions in Boxes 2 and 4, respectively, by "$I < 50$", and "$K \leq J$ and $I + J + K \leq 100$".

Comment on the solution to Problem 7

One drawback to the given problem as an example of a computer problem is that formulas for the solutions can be generated by elementary techniques. Thus, in part (a) if we replace 100 by 2M in order to generalize we have

$$S = \sum_{I=1}^{2M} \; \sum_{J=1+[I/2]}^{I} (2J-I)$$

$$= \sum_{\substack{I=1 \\ I \text{ odd}}}^{2M} \; \sum_{J=1+[I/2]}^{I} (2J-I) + \sum_{\substack{I=1 \\ I \text{ even}}}^{2M} \; \sum_{J=1+[I/2]}^{I} (2J-I)$$

$$= \sum_{H=1}^{M} \; \sum_{J=H}^{2H-1} (2J - 2H+1) + \sum_{H=1}^{M} \; \sum_{J=H+1}^{2H} (2J - 2H)$$

$$= \sum_{H=1}^{M} (2H^2 + H) = 2\frac{M(M+1)(2M+1)}{6} + \frac{M(M+1)}{2}$$

$$= \frac{M(M+1)(4M+5)}{6}$$

which has the value 87,125 for $M = 50$ and this is the answer to part (a). For the other parts we obtain the result by similar but slightly more difficult computations. In Chapter 5 we return to this problem, replacing the word "distinct" by "dissimilar" (i.e., no two similar). This slight modification yields a problem no more difficult for a machine, but it is no longer so easily accessible by means of deriving a formula.

Answer to Exercise 4-4 Set C

The student is entirely correct in both his claims. Students may have a little trouble following Figure 4-33. The trouble spot students may encounter is accepting the concept that

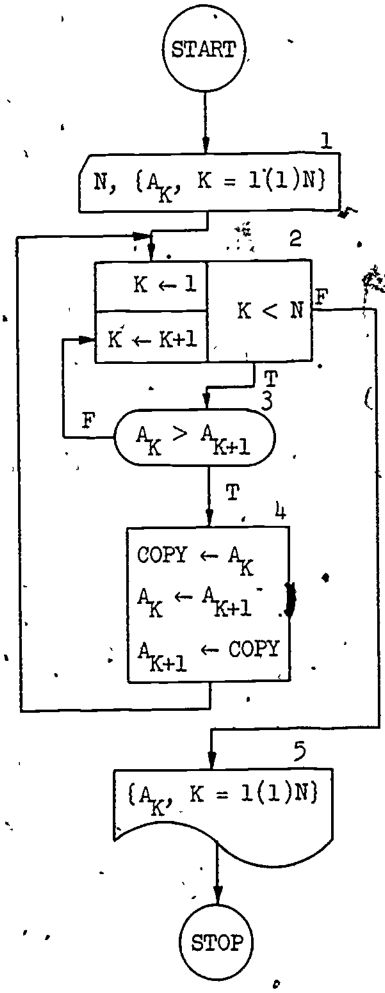
$$K \leftarrow K$$

is a legitimate initialization for Box 2. It is valid and is a useful "trick" in this particular version of the algorithm.

The algorithm in Figure 4-33 is equivalent to the one in Figure 4-32 and is more efficient in that unnecessary repetition of the computation of $\sqrt{N}$ is avoided.

Answers to Exercises 4-4 Set D

1.



Check values.

Input N = 6

A_1	A_2	A_3	A_4	A_5	A_6
4	6	1	3	5	2

Output

1, 2, 3, 4, 5, 6

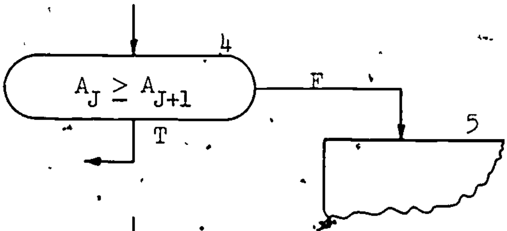
2(a) 5 times (3 times for Box 4 and 2 times for Box 7). You should urge the students to trace through Figure 4-34 until they can get the correct answer for this or for some other set of data. Students who have difficulty should be asked to review the tracing techniques which they used for Exercise 1(b) Section 3-5 Set B.

(b) 3 times (3 times for Box 4 and none for Box 7).

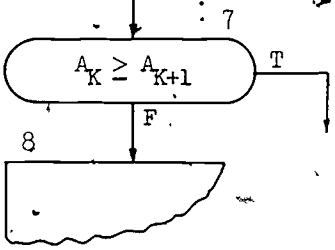
(c) 6 times (3 times for Box 4 and 3 times for Box 7)

In general, for N numbers initially in reverse order, $\frac{N \times (N - 1)}{2}$ comparisons are required. This is for the worst case. It is far more efficient than the primitive sort for which $N^3 + 5N - 6$ comparisons were required. In the most favorable case (where the data is already completely sorted) both algorithms require N - 1 comparisons.

3. Box 4 should be



Box 7 should be



No other changes are needed.

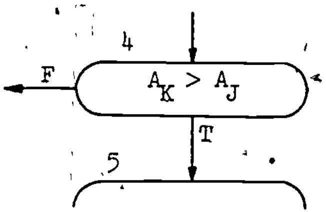
4. Claim (a) is correct. This is a perfectly good algorithm for sorting in ascending order. (It's been called the "push down" method because the first largest number is pushed down to the bottom of the list, then the next largest is pushed down to a point just above the largest, and so forth.)

Claim (b) is false. The push down method requires $\frac{N \times (N - 1)}{2}$ comparisons regardless of the initial ordering exhibited by the data. Algorithm 4-34 requires this number of comparisons only in the worst case (reverse order input data).

Answers to Exercises 4-4 Set E

1. (a) Change all B's to C's and MAXINC's to MAXDEC's.

   (b) Change Box 4 to read:



2. Either $A_J < A_K$ or $A_J > A_K$.

   If $A_J < A_K$, then A_K can be tacked on the end of the longest monotone subsequence ending with A_J. Thus, $B_K > B_J$. Similarly, if $A_J > A_K$, then $C_K > C_J$.

3. From Problem 2 we see that for each integer I from 1 to N there is a corresponding pair

$$(B_I, C_I),$$

and no two pairs are the same. Let M be the length of the longest monotone subsequence. Then each of the B_I and the C_I is less than or equal to M. The number of possible different pairs (B_I, C_I) is then M^2. Since we have N such pairs it must be that $N \leq M^2$ (i.e., that $M \geq \sqrt{N}$).

A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
7	8	9	4	5	6	1	2	3

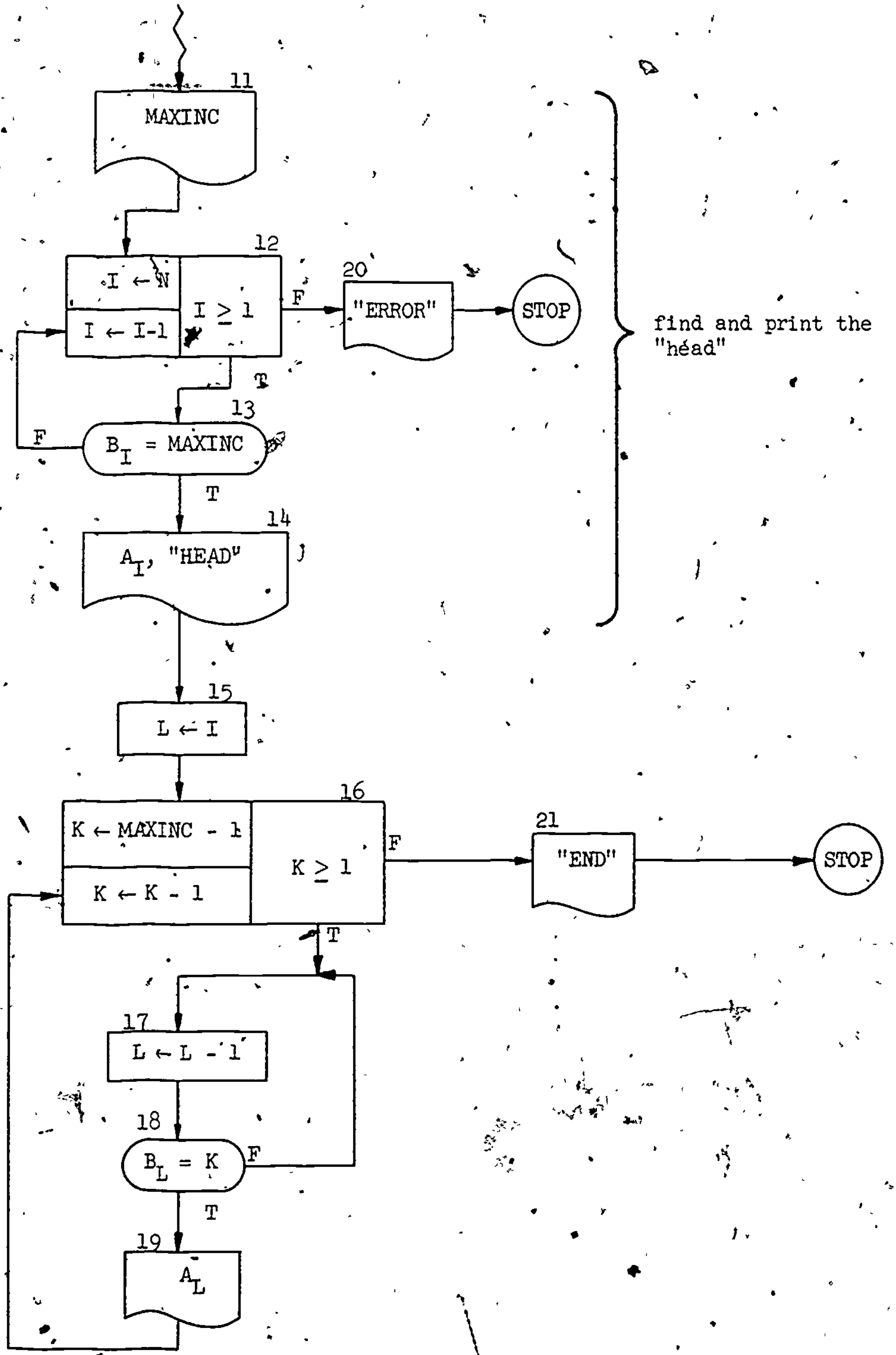
indicates how we may, for any value of N, construct sequences having no monotone subsequences of length as great as $1 + \sqrt{N}$. Thus, we cannot increase the lower bound of $\sqrt{N}$ for the length of the largest monotone subsequence.

4. The job of finding a maximum subsequence consists of carrying out these two steps.

   (1) Search backwards from B_N looking for the first value, B_K, which is equal to MAXINC. The corresponding value of A_K is the head of subsequence. Print out this value A_K.

   (2) Now resume the backward search of the B's for the first one whose value is one less than that of the previous B_K and print the value of the corresponding element in the A vector.

   Repeat the process of searching backwards for successively smaller values of B_K, printing out the corresponding value of A_K. The last value of A to be printed will be the one for which the value of B_K is one. Here is the flow chart.



Sample Test Questions

The following questions do not constitute a test. They are offered to the teacher as typical or appropriate sample test items. In some cases a single item would be enough for one test. The chapter designation on a question means that the item can be used any time after the completion of the indicated chapter.

1. (Chapter 2) Given the length (a, b, and c) of the three sides of a triangle, the area can be determined by Heron's formula:

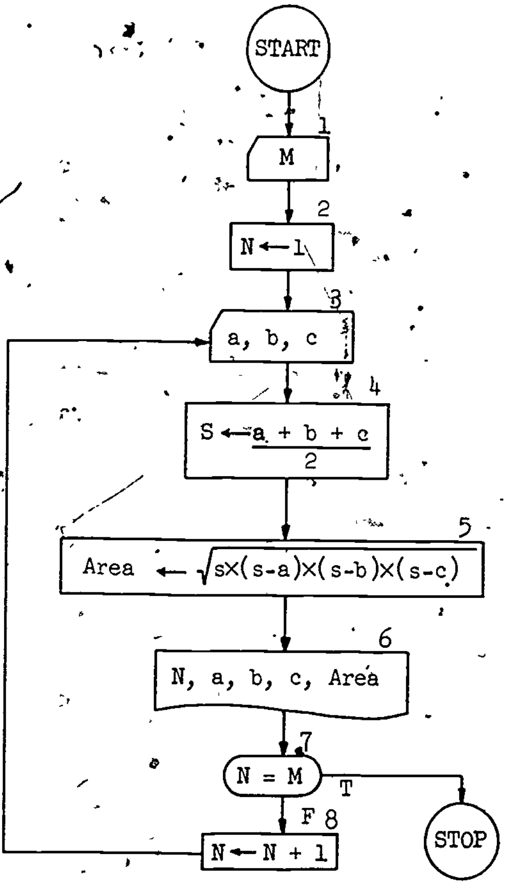
$$\text{Area} = \sqrt{s(s - a)(s - b)(s - c)}$$

where

$$s = \frac{a + b + c}{2} .$$

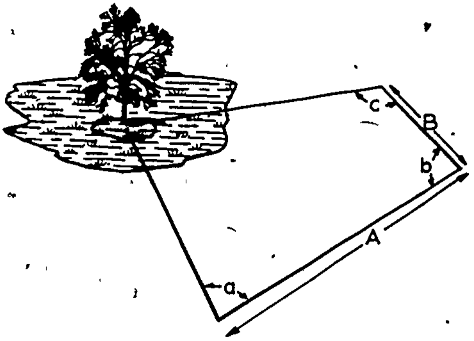
Draw a flow chart which will compute M such areas for given lengths of the sides and then stop. Assume you have M data cards, each containing three positive numbers representing the sides of one triangle. Each line of output should contain the count, N, the three lengths, and the area.

Solution:



2. (Chapter 3)

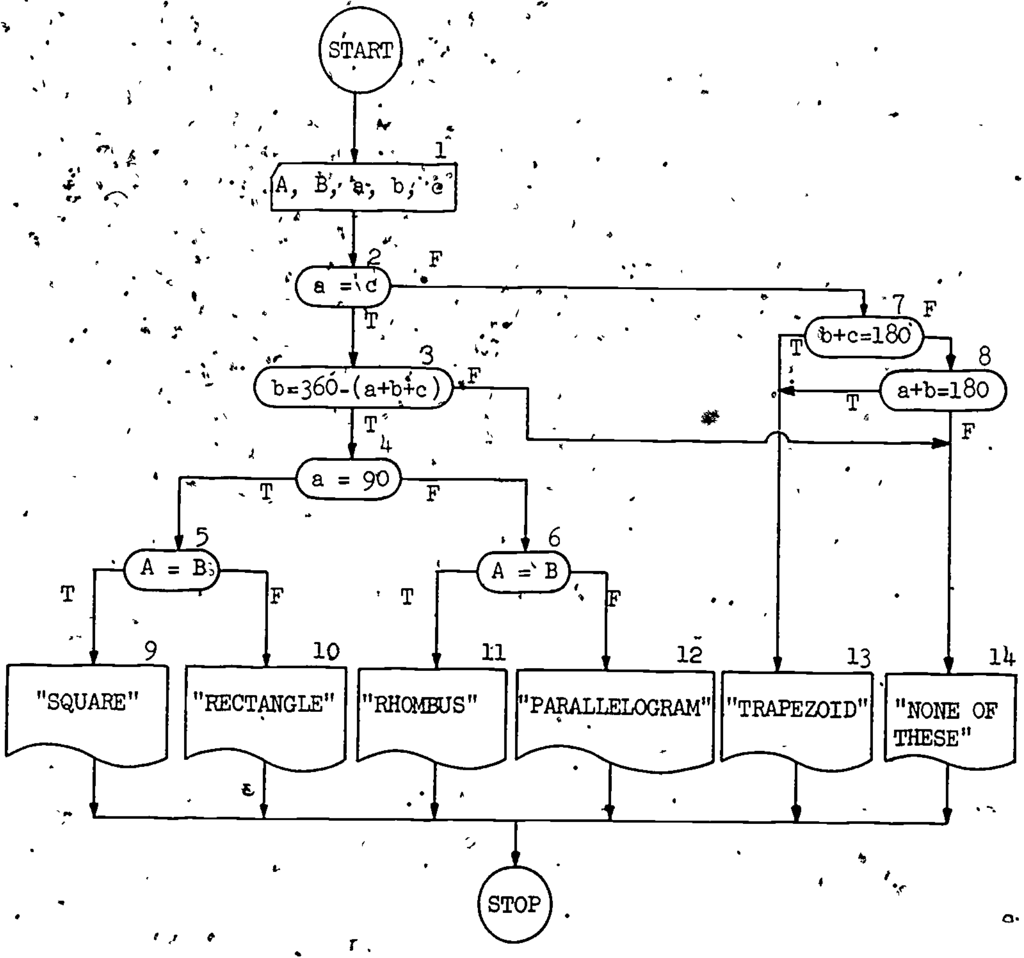
A surveyor measures the sides and angles (in degrees) of a quadrilateral as shown on the figure below. Construct a flow chart to decide whether the quadrilateral is:



1. a square;
2. a rectangle (but not a square);
3. a rhombus (but not a square or a rectangle);
4. a parallelogram (but not a 1, 2, or 3);
5. a trapezoid (but not a 1, 2, 3, or 4);
6, none of these.

Output an appropriate message in each case.

Solution:



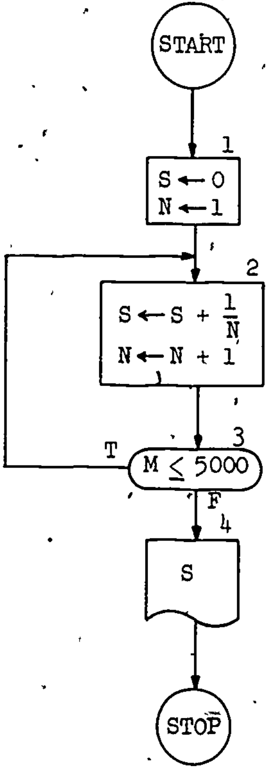
3. (Chapter 3 or Chapter 4)

a. Draw a flow chart for computing the sum of the reciprocals of the first 5000 positive integers.

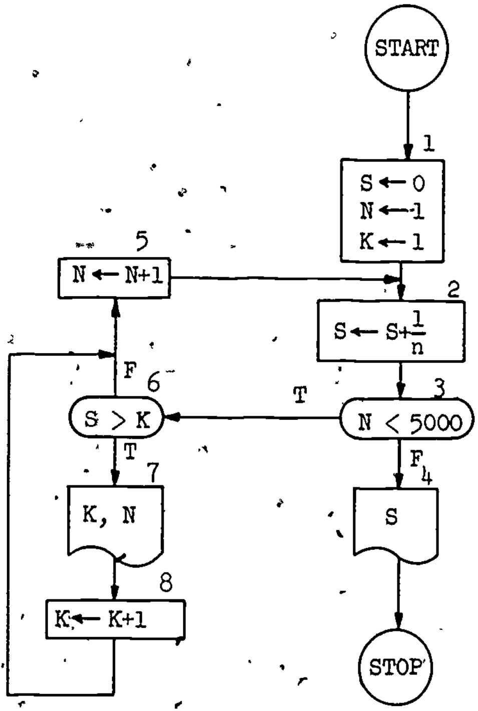
b. Add to this flow chart a mechanism for printing out the number of terms used when the sum first exceeds 1, first exceeds 2, etc.

Solution: (If given following Chapter 3)

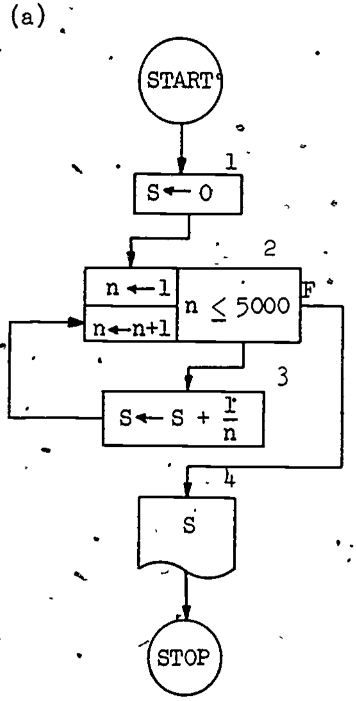
(a)

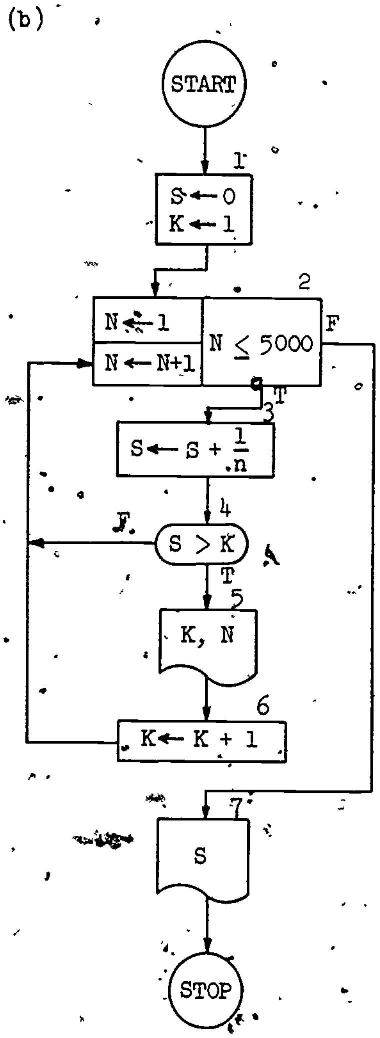


(b)



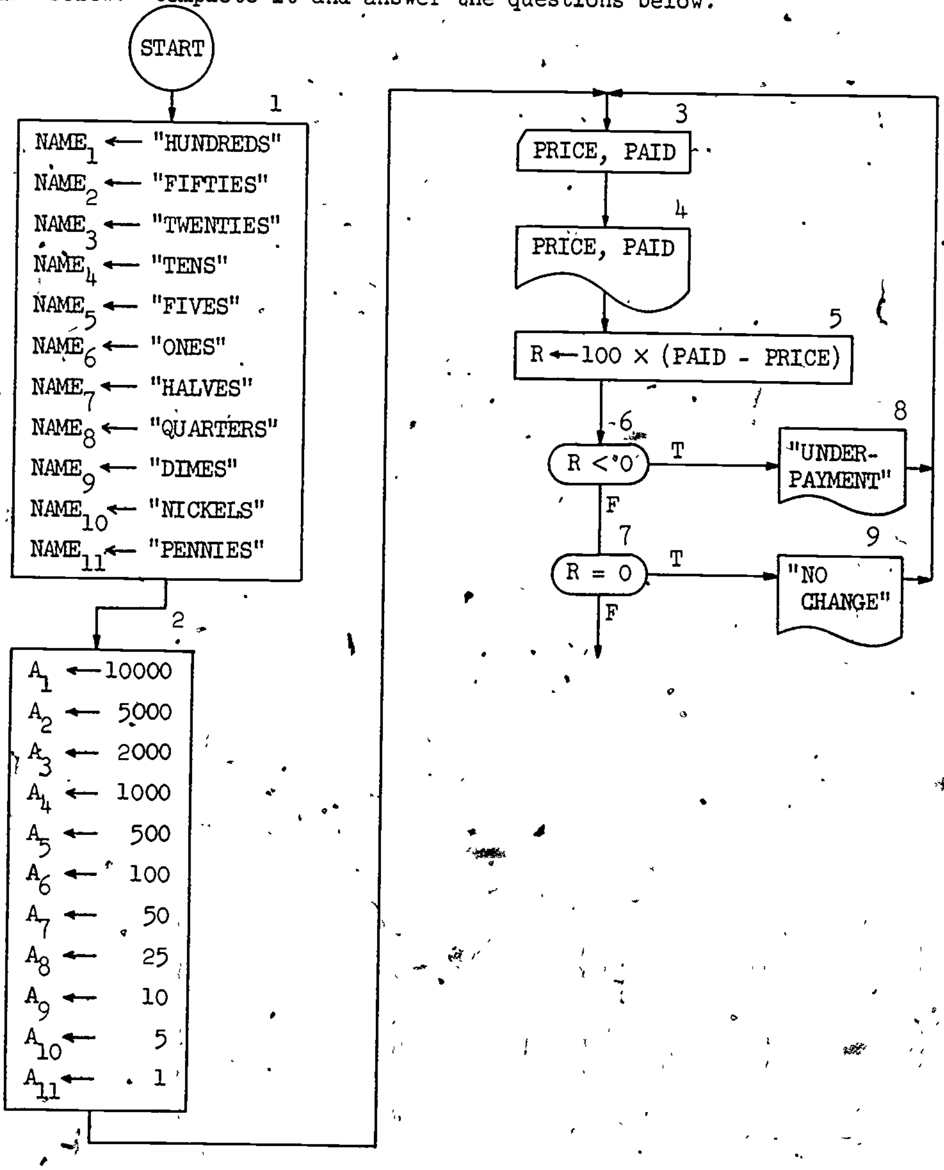
(Note to teacher: If given following Chapter 4, iteration boxes would be used as shown below.)





4. (Chapter 4)

Most of us often have to "make change". The problem is to draw a flow chart for making change in the fewest number of bills and coins. You should output the names of the bills and coins actually appearing in the change together with the number of each. Then loop back to make change for the next transaction. We have started you off in the partial flow chart below. Complete it and answer the questions below.

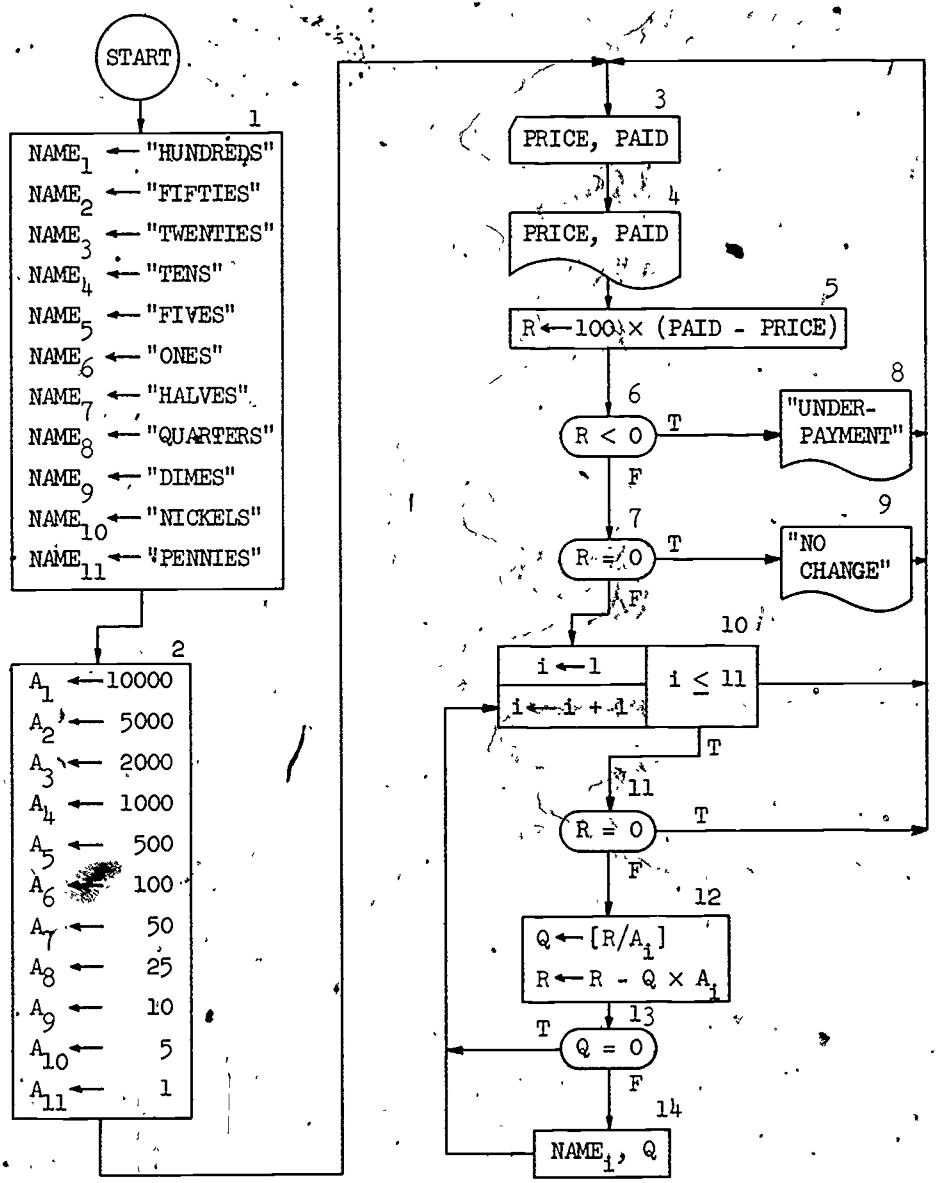


a. Explain the connection between $NAME_i$ and A_i.

b. Why is PAID - PRICE multiplied by 100 in Box 5 ?

c. In what form should the data referred to in Box 3 be input? Give an example.

Solution: The students' contribution consists of Boxes 10 - 14 and their connections.



a) A_i is the equivalent in pennies of the coin or bill called $NAME_i$.

b) To convert the change into pennies.

c) In dollar and cents; e.g., PRICE = 3.49; PAID = 10.00.

[Note to teacher: Box 11 could be eliminated entirely or it could be put in after Box 14. Including this test increases the efficiency of the algorithm.]

5. (Chapter 4)

For the accompanying flow chart:

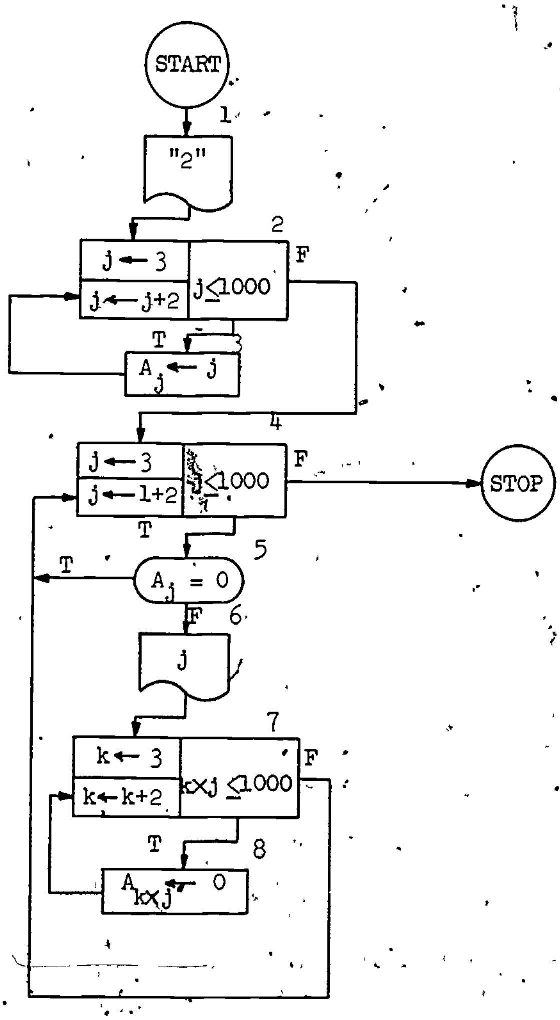
i) Describe in your own words the values which j will successively be assigned in Box 4.

ii) Note that for each odd number j, A_j is initialized with the value j in Box 3. Supposing that we leave Box 6 with j having a particular value, for what values of i will A_i be crossed out (i.e., set equal to zero) in the ensuing loop through Boxes 7 and 8 ?

iii) Describe in terms of "crossed out" the circumstances under which we leave Box 5 by the T exit.

iv) Give the first 7 output values. Use scratch paper if necessary.

v) Describe what the algorithm is doing, in your own words.



Solution:

i) j will be assigned successively the odd numbers starting with 3, i.e., 3, 5, 7, 9, etc. As long as these values are ≤ 1000, the T exit is taken to Box 5. The last value for which this happens is $j = 999$. The next time $j = 1001$ and $j \leq 1000$ is false, so the F exit is taken.

ii) All the odd multiples of j less than a thousand will be crossed out, i.e., $3 \times j$, $5 \times j$, $7 \times j$, etc.

iii) If $A_j = 0$, then it has been "crossed out" and is not a candidate to be output. Any odd number which can be "reached" as an odd multiple of a smaller odd number will get "crossed out". We will leave Box 5 by the F exit only when j is a prime. (Since there are no even primes greater than 2, none are missed by excluding even numbers from consideration.)

iv) 2, 3, 5, 7, 11, 13, 17.

v) The algorithm is generating the primes less than one thousand. In Box 1, "2" is output as a special case, the only even prime. Then a loop assigns the odd numbers as the components of a vector with the same subscripts, $A_j \leftarrow j$. Then the components are tested one at a time starting with 3. If the component is not zero, then it is a prime. Not only is a prime output in Box 6, but all its odd multiples are crossed out because they are composite numbers. Since the even multiples of odd numbers are even, and hence composite, only odd multiples need be considered in Boxes 7 and 8.

6. (Chapter 4)

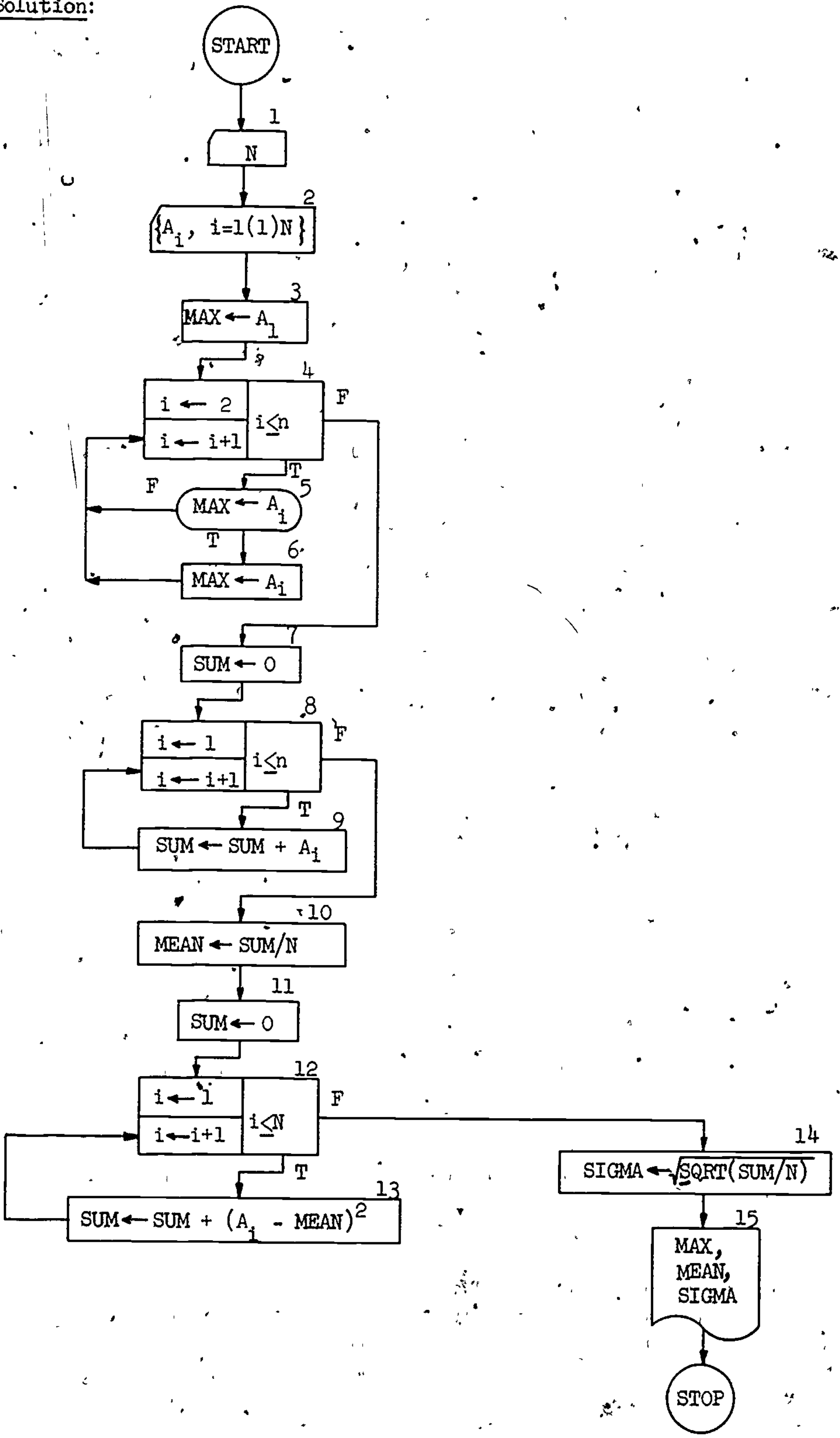
Construct one flow chart to do all the following:

1. read in a number N;
2. read in an array A which has N components;
3. compute MAX, the largest of the components;
4. compute MEAN, the arithmetic mean of the numbers A_i;
5. compute the standard deviation SIGMA of the set of numbers A_i;

$$\text{SIGMA} = \sqrt{\frac{1}{N}\left(\sum_{i=1}^{N} (A_i - \text{MEAN})^2\right)}$$

6. Write MAX, MEAN, and SIGMA.

Solution:


START
1
N
2
{A_i, i=1(1)N}
3
MAX ← A_1
4
i ← 2
i ← i+1
i≤n
F
T
5
F
MAX ← A_i
T
6
MAX ← A_i
7
SUM ← 0
8
i ← 1
i ← i+1
i≤n
F
T
9
SUM ← SUM + A_i
10
MEAN ← SUM/N
11
SUM ← 0
12
i ← 1
i ← i+1
i≤N
F
T
13
SUM ← SUM + (A_i - MEAN)^2
14
SIGMA ← √SQRT(SUM/N)
15
MAX,
MEAN,
SIGMA
STOP

Chapter T5

FUNCTIONS AND PROCEDURES

Overview

One can use the same sequence of boxes in several unrelated parts of one flow chart (or several flow charts), by writing a *reference flow chart* once, and referring back to it in the several places. Section 5-1 explains what these reference flow charts are and how to prepare them.

Section 5-2 is a review discussion of the function concept from a mathematical point of view followed by remarks on the function concept in computing.

Two slightly different types of reference flow charts are possible. The first is called a *functional* reference flow chart because the end result is the computation of a single value, just as in applying a function, a single value resulted. This type of flow chart is described in Section 5-3.

The second type of reference flow chart is more general and is called a *procedure*. Section 5-4 explains the convention we will use for procedures.

Section 5-5 explains how to include alternate exits in procedures. A procedure can also be used to make complicated comparisons or evaluations for use as branching criteria. Section 5-5 shows how to do this.

The last section of the chapter, 5-6, develops several procedures using character strings. These have double importance. They illustrate how one procedure can use another and thus show the building block property of procedures. Secondly the procedures themselves will be used again in Chapter 8.

5-1 Reference Flow Charts

The point of this section is to introduce the idea of producing flow charts that may be referred to many times from the same point or from many different points in another flow chart. Newton's algorithm for the square root is used as a means of bringing up this idea with the associated flow chart conventions.

Of course, the square root of A is a solution of

$$f(x) = x^2 - A = 0.$$

In calculus the student will learn that $f(x)$ can be approximated near $x = x_0$ by a series:

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0) + \ldots$$

where $f'(x_0)$ is the derivative of the expression $f(x)$ evaluated when $x = x_0$. If we take only this first (linear) term as a suitable approximation,

$$x^2 - A \cong x_0^2 - A + (x - x_0)2x_0 .$$

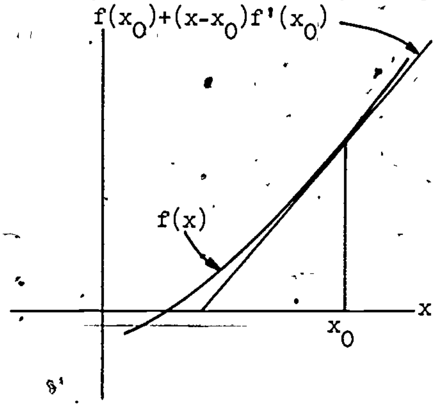
If x_0 represents our first guess at the square root of A, the x determined by setting the above approximation to zero would be closer to the square root of A. Then:

$$(x - x_0) = \frac{-x_0^2 + A}{2x_0}$$

$$x = \frac{2x_0^2 - x_0^2 + A}{2x_0} = \frac{1}{2}(x_0 + \frac{A}{x_0}).$$

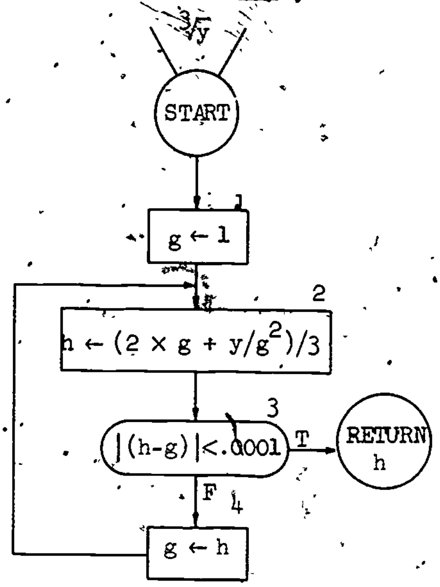
The same argument for $f(x) = x^3 - A$ or $f(x) = x^N - A$ produces the formulas for Exercises 5-1 in this section and Problem 6 in Exercises 5-3, Set A.

Now, what we are really doing by saying that $f(x) \cong f(x_0) + (x - x_0)f'(x_0)$ is to approximate the graph of $f(x)$ by a straight line with slope $f'(x_0)$ and touching the graph of $f(x)$ at $x = x_0$. Then the root of the line (its intersection with the x-axis) is an approximation to the root of $f(x)$. Newton's method is discussed in more detail in Section T7-1 in connection with the procedure ZERO.

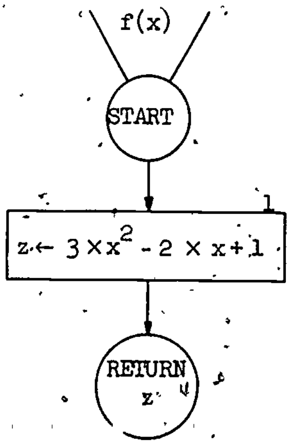


Answers to Exercises 5-1

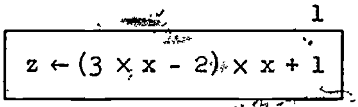
1.



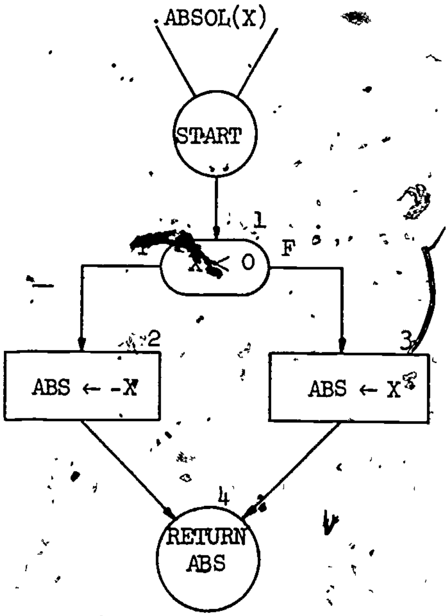
2.



Alternatively:



3.



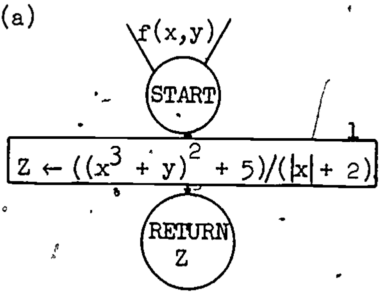
Answers to Exercises 5-3 Set A

Discussion

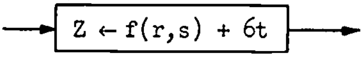
It is important that the student develop the habit of recognizing and repairing all "loopholes" in his flow charts. He should not just assume that input data will be reasonable; this is especially important for a function or procedure since not only is it to be used with many different programs, but also the input data may itself be generated within the main program. He must be alert to the possibility of dividing by zero or taking the square root of a negative quantity. In this exercises and later ones, possible sources of error should be pointed out to him if he has not already seen them. Then a more formal discussion of alternate (error) exits will be given in Section 5-5.

Solutions

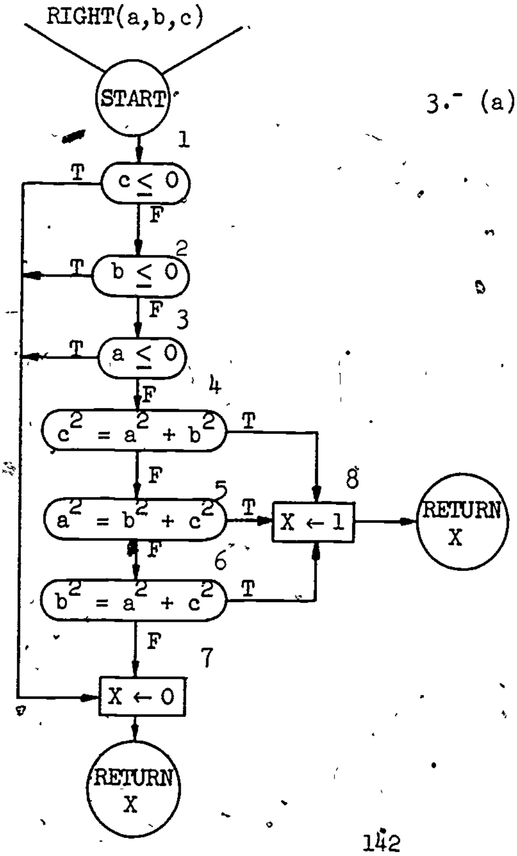
1. (a)



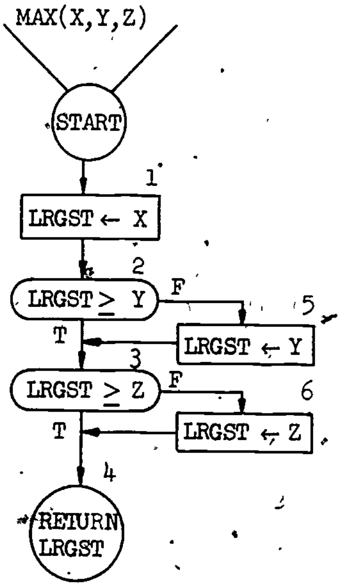
(b)



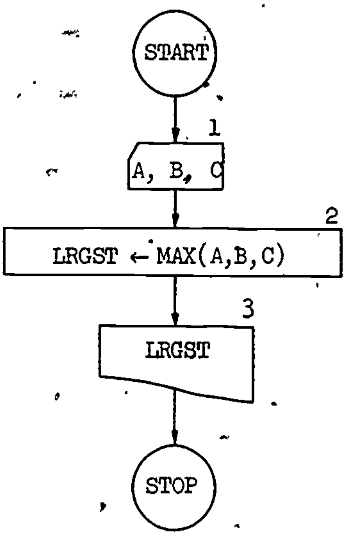
2.



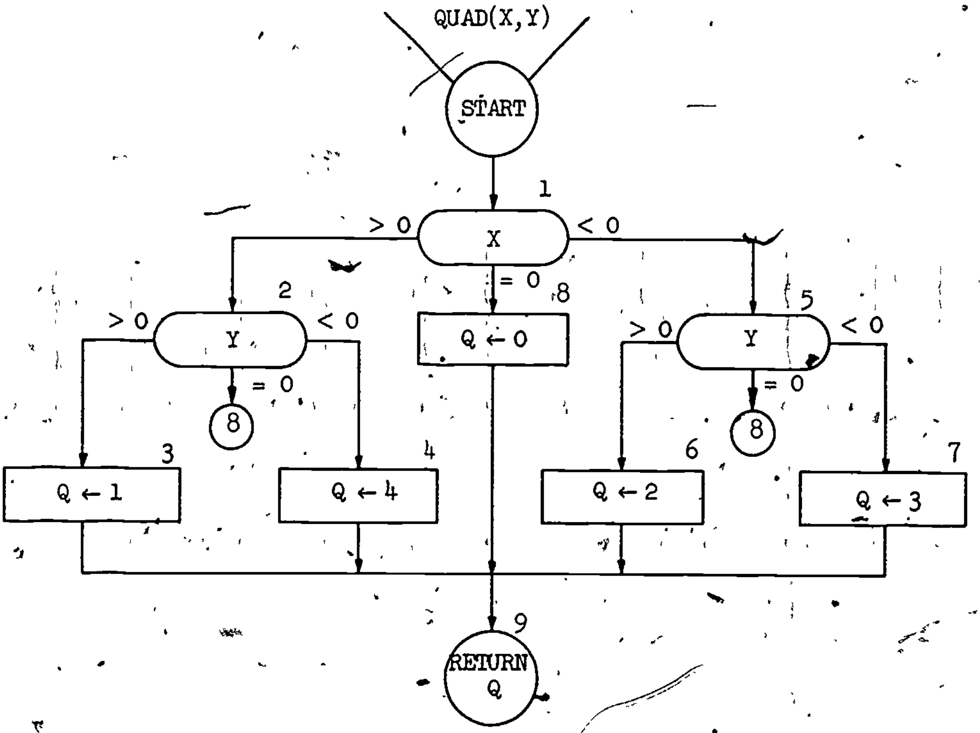
3. (a)



3. (b)



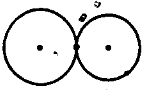
4. Here as an error indication we set Q equal to zero if the point is on a coordinate axis.



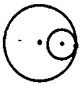
5. Defining dist to be the distance between centers, we have

for dist = 0, concentric circles with $\begin{cases} \text{infinite points of intersection for } R1 = R2 \\ 0 \text{ points of intersection for } R1 \neq R2 \end{cases}$

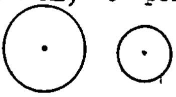
for dist = R1 + R2, 1 point of intersection



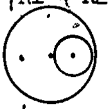
for dist = |R1 - R2|, 1 point of intersection



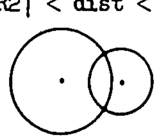
for dist > R1 + R2, 0 points of intersection



for dist < |R1 - R2|, 0 points of intersection

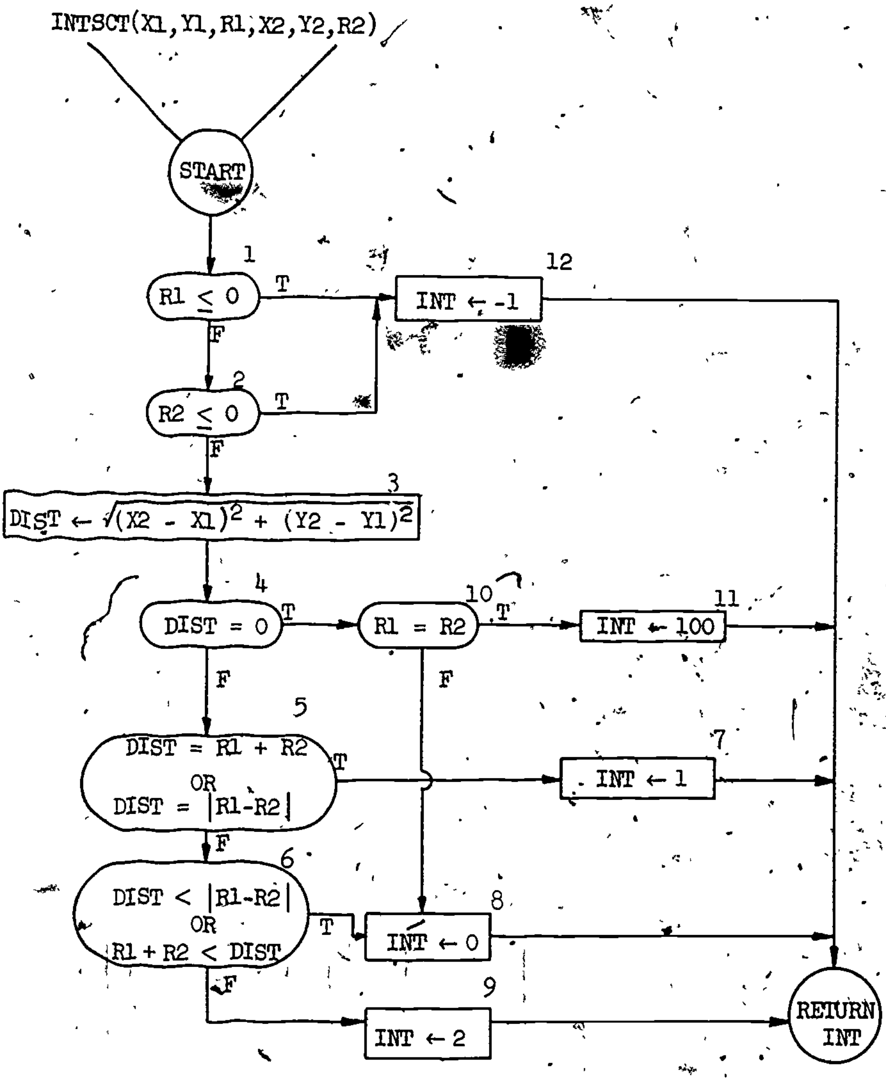


for |R1 - R2| < dist < R1 + R2, 2 points of intersection

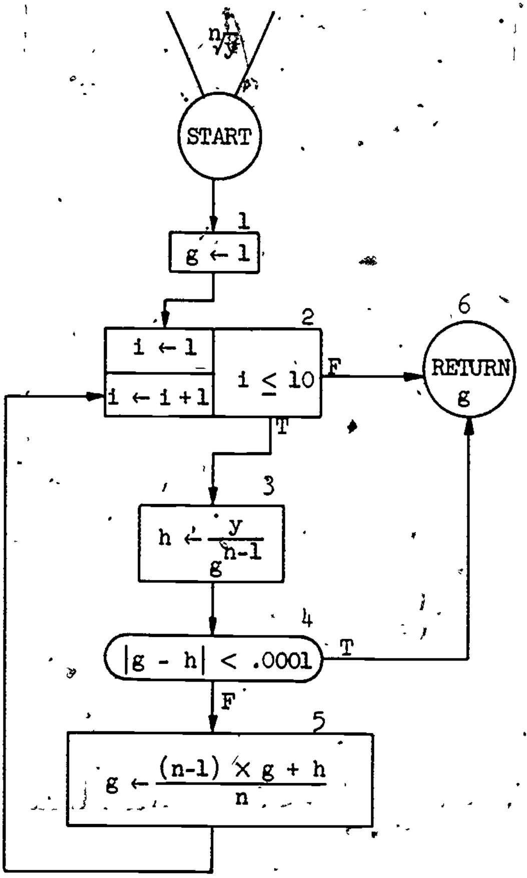


Also, we must check the data to make certain R1 and R2 are positive; an error is indicated by INT = -1. Concurrent circles are indicated by INT = 100.

5. (continued)


INTSCT(X1,Y1,R1,X2,Y2,R2)
START
1
R1 ≤ 0
T
F
12
INT ← -1
2
R2 ≤ 0
T
F
3
DIST ← √(X2 - X1)² + (Y2 - Y1)²
4
DIST = 0
T
F
10
R1 = R2
T
F
11
INT ← 100
5
DIST = R1 + R2
OR
DIST = |R1-R2|
T
F
7
INT ← 1
6
DIST < |R1-R2|
OR
R1 + R2 < DIST
T
F
8
INT ← 0
9
INT ← 2
RETURN
INT

6.



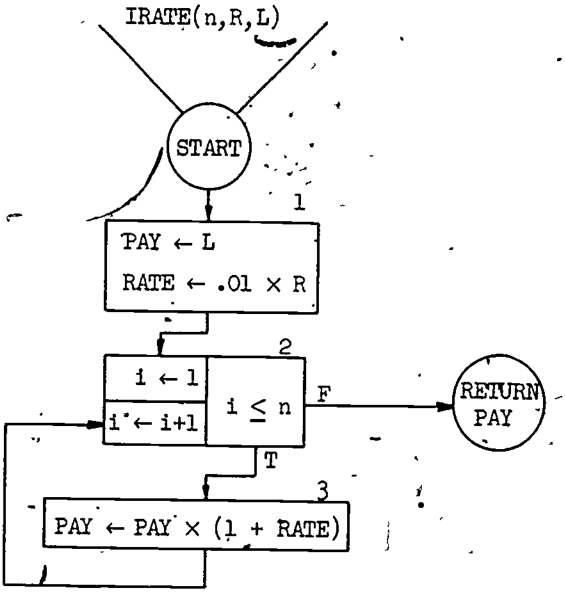
Comment: Here h is used differently than in Figure 5-7. We have let h designate the other end of the interval in which the root is known to be bracketed. Thus, if the test is passed in box 4, we <u>know</u> that the root is within .0001 of g.

7. We use this problem to emphasize two points.

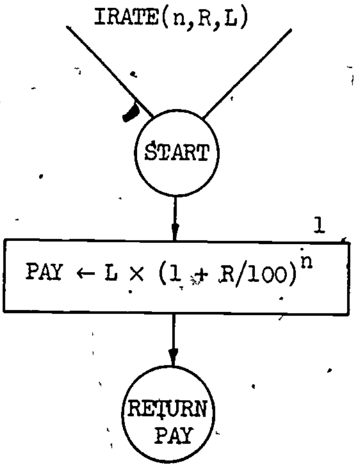
(1) There are always alternate solutions and generally various advantages and disadvantages to be balanced before anyone can say which is better.

(2) Our general approach is to build up a set of tools (subroutines) and use them widely. Problem 7(b) illustrates a case where we are better off not using a subroutine that is already available.

(a)

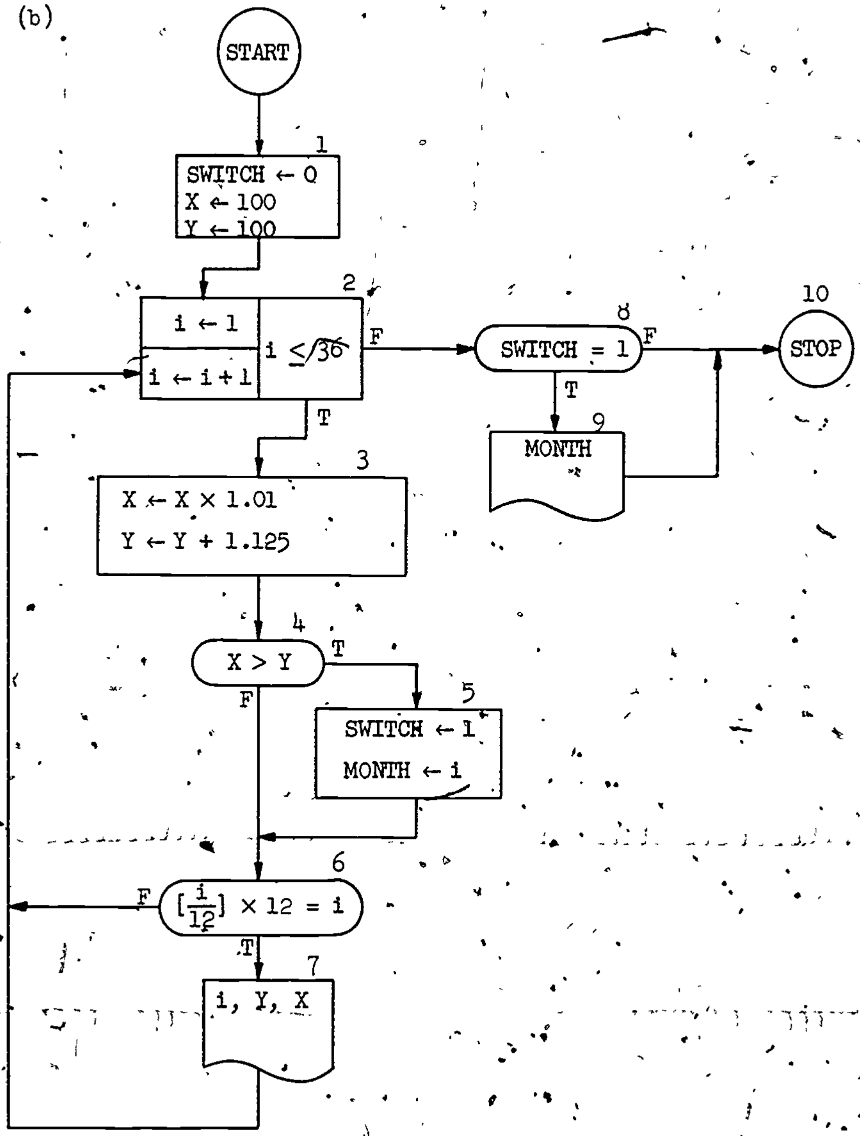


alternate solution:

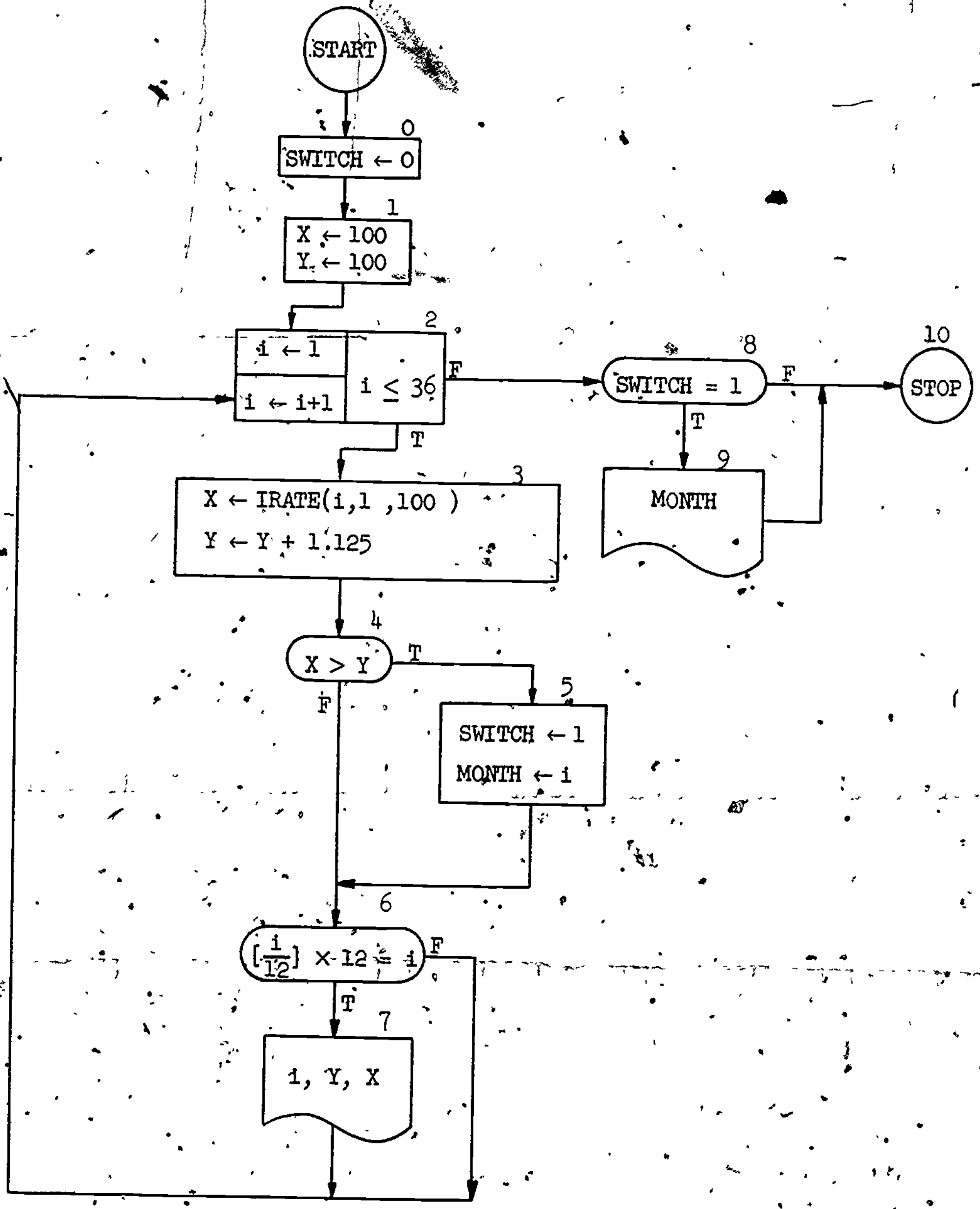


Comment: The alternate solution looks much simpler but $n + 1$ multiplications are still needed.

7. (b)


START
1
SWITCH ← 0
X ← 100
Y ← 100
2
i ← 1
i ← i + 1
i ≤ 36
F
T
3
X ← X × 1.01
Y ← Y + 1.125
4
X > Y
T
F
5
SWITCH ← 1
MONTH ← i
6
$[\frac{i}{12}] \times 12 = i$
F
T
7
i, Y, X
8
SWITCH = 1
F
T
9
MONTH
10
STOP

7. (b) Alternate solution:



Comment: This is considered an undesirable solution because IRATE is called 36 times, each time taking longer to compute. (Total number of passes through box 3 of IRATE flow chart is $\frac{36 \cdot 37}{2} = 665$ as compared with 36 passes through box 3 of the first solution.)

7. (c) For company X

$$200 = 100 \times (1.01)^n$$

$$2 = 1.01^n$$

$$\log 2 = n \times \log 1.01$$

$$n = \frac{\log 2}{\log\ 1.01}$$

more precisely

$$NX = ROUNDUP(n) = ROUNDUP(\log 2/\log 1.01)$$
$$= -[-\log 2/\log 1.01]$$

For company Y

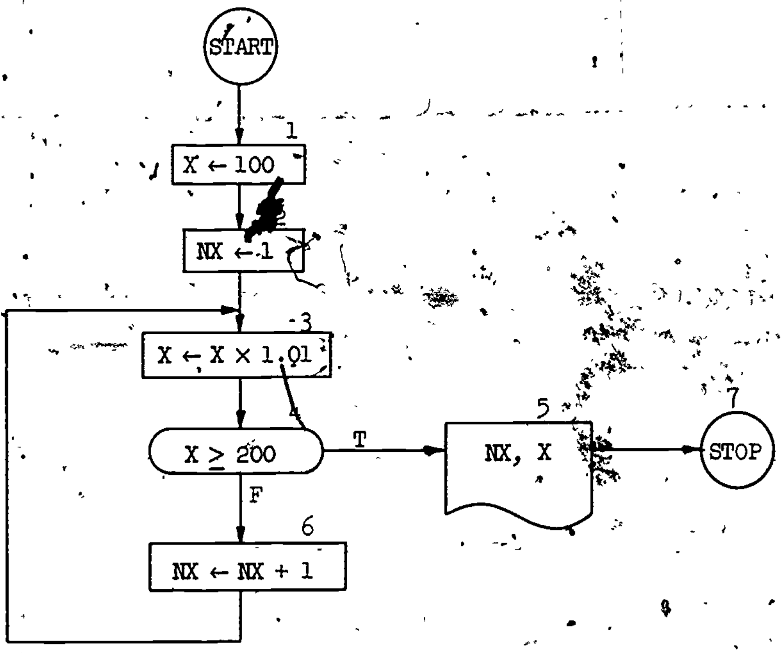
$$200 = 100 + 1.125 \times n$$

$$100 = 1.125 \times n$$

$$n = \frac{100}{1.125}$$

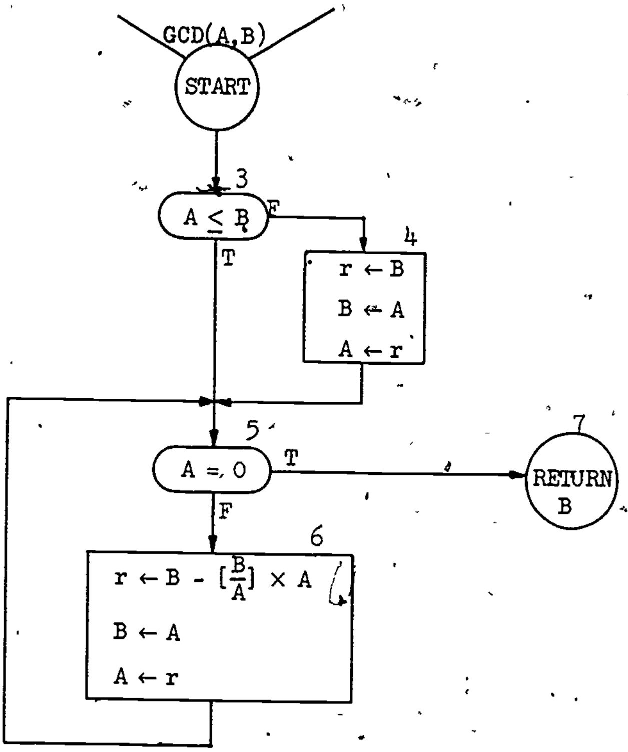
$$NY = ROUNDUP(n) = -[- 100/1.125]$$

Alternate solution for NX (not requiring logarithms):



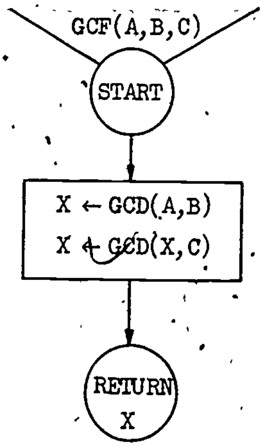
Answers to Exercises 5-3 Set B

1.

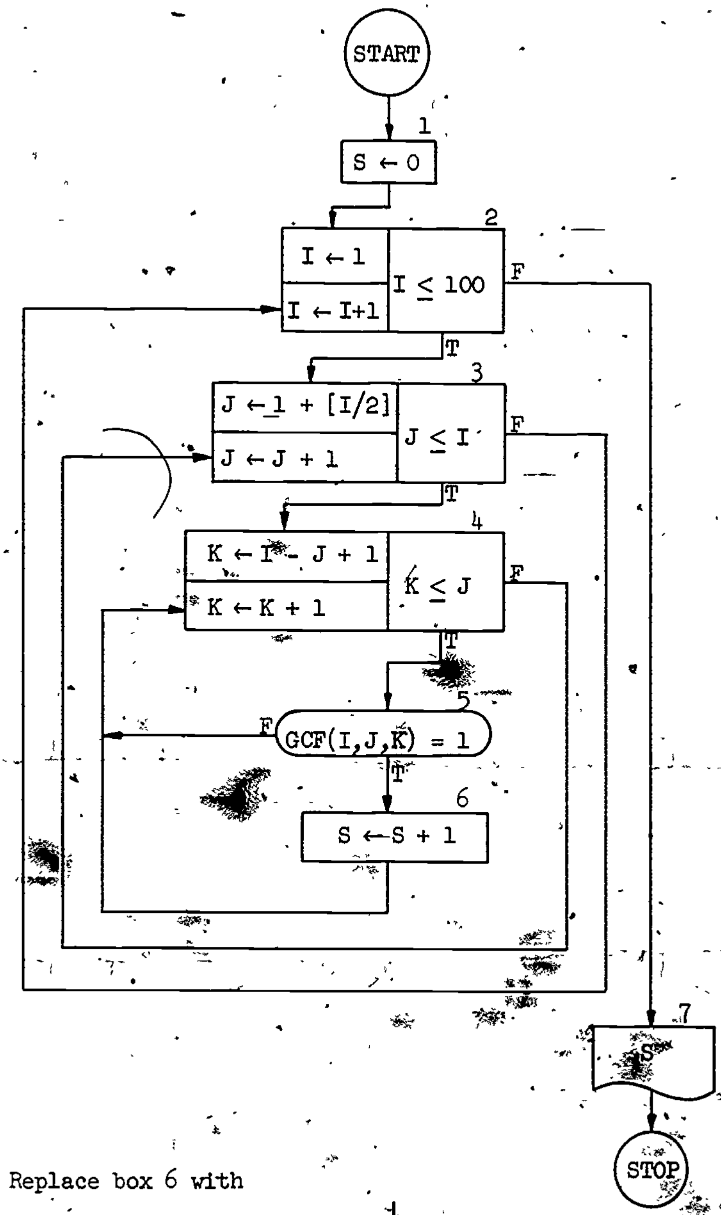


We replace box 1 with the funnel, box 7 with the return box and omit box 2 entirely. It's unnecessary for a functional reference flow chart.

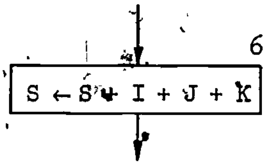
2.



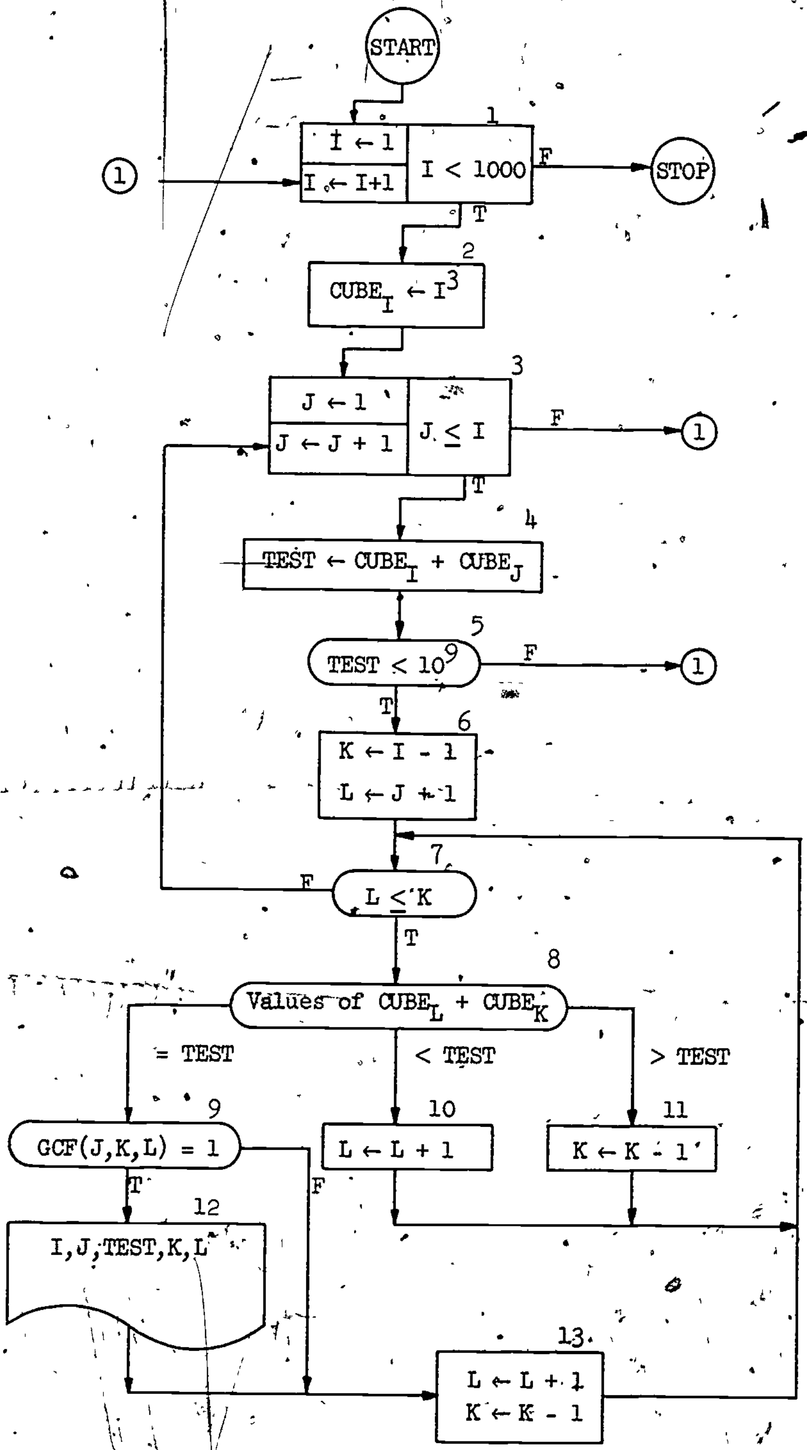
3. (a) (See remarks in the TC in Section 4-4, Set B, Problem 7.)



(b) Replace box 6 with



4. We give a flow chart for the solution and then provide some discussion.



Discussion: Box 2 together with box 1 shows us that storage locations will be needed for all perfect cubes between 1^3 and 1000^3, that's 1000 in all. The only other storage locations needed will be those for the variables I, J, TEST, K and L. We also need the reference flow chart for GCF (box 9). The number of passages through box 7 is approximately $10^9/6$ and the total number of arithmetic operations will be on the order of $2\cdot10^9$, a formidable number! These large storage and time demands may be scaled down by replacing 10^9 in the statement of the problem by 10^6. Then only 100 storage locations for cubes will be needed and the number of passes through box 7 will be reduced to $10^6/6$ and the number of arithmetic operations to $2\cdot10^6$. Moreover, word length requirements will be reduced from 8 decimal digits to 5. The only changes in the flow chart will be replacing 1000 in box 1 to 100 and 10^9 in box 5 to 10^6.

An interesting feature of the flow chart is that not all the 10^9 numbers from 1 to 10^9 are tested to determine whether they meet the conditions of the problem, but only those (TEST in box 4) which already are known to be the sum of two cubes in at least one way. They are approximately $10^6/2$ in number. We are looking for integers I, J, K and L so that

$$I^3 + J^3 = K^3 + L^3$$

where I is the largest of these four numbers and K the second largest, whence it follows that J is the smallest and L the second smallest. These observations are reflected in boxes 6 and 7 and in the test in box 3.

The "see-saw" technique displayed in boxes 10, 11 and 13 will be appreciated after careful study. (If $J^3 + L^3$ is too small, increase L, but if too large, decrease L.) Box 9 eliminates proportional combinations. If three of the numbers I, J, K and L have a common factor, then the fourth must also have this factor since $I^3 + J^3 = \text{TEST} = K^3 + L^3$.

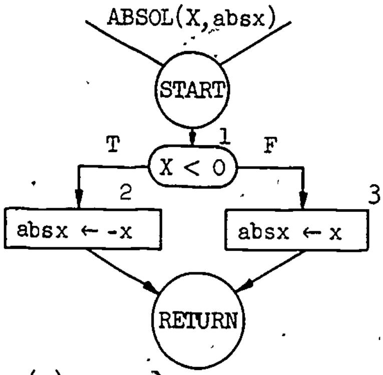
The modifications required to find the numbers which are the sums of two squares (or fourth powers or fifth powers) in two different ways are trivial. But, in the case of squares, the limits must be scaled down to avoid the use of tons of paper.

Hooking box 12 into box 13 instead of box 3 assures that if any numbers can be expressed as the sum of cubes in more than two ways, we will find this out.

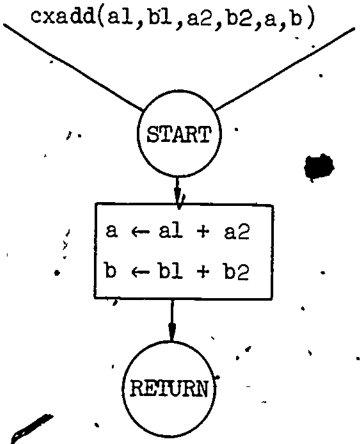
Answers to Exercises 5-4 Set A

Discussion: A reasonable selection could be problems 1, 3 and 4 with the corresponding problems in the language text. On considering problem 5, you might wish to assign only parts (a) and (b); (c) is a rather tricky challenge for the better student.

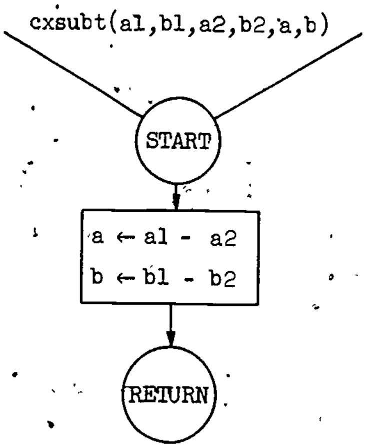
1.



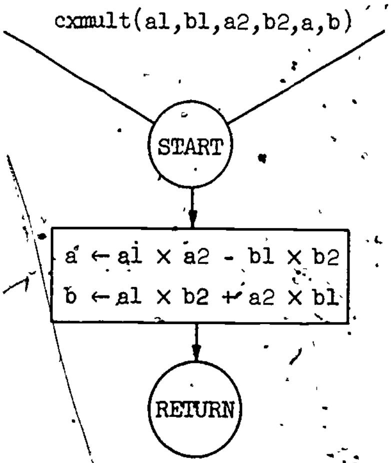
2. (a)



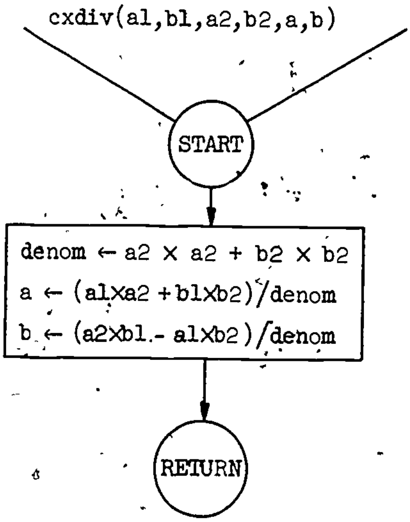
(b)



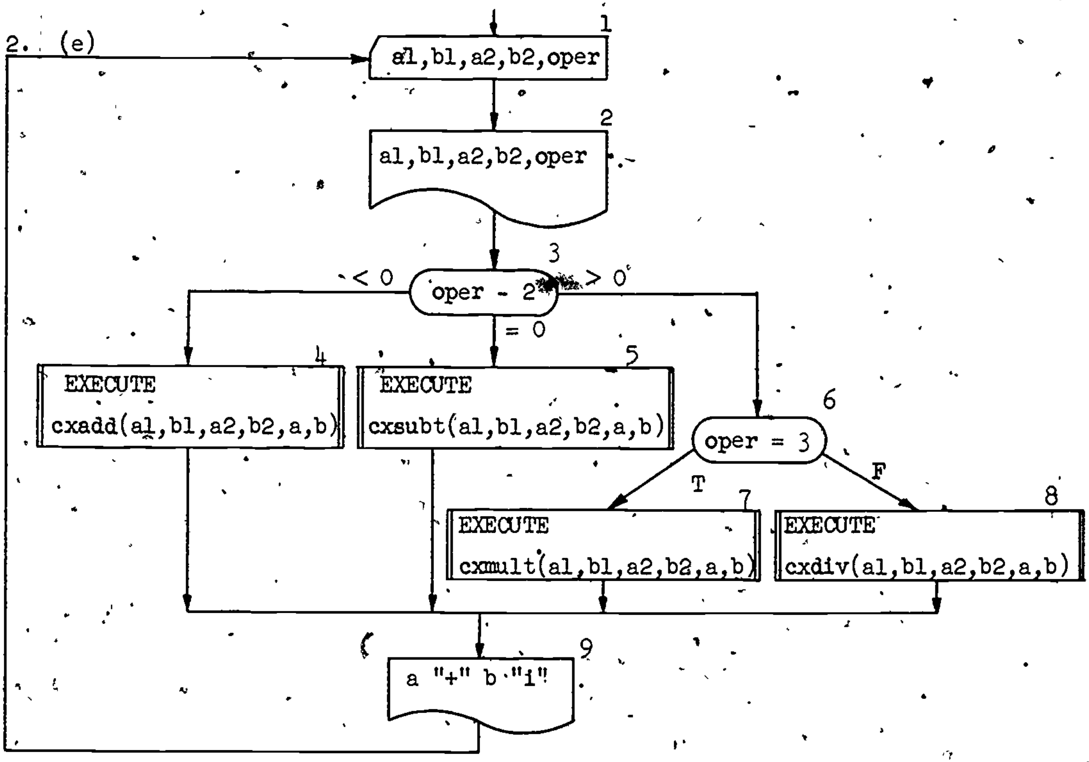
(c)



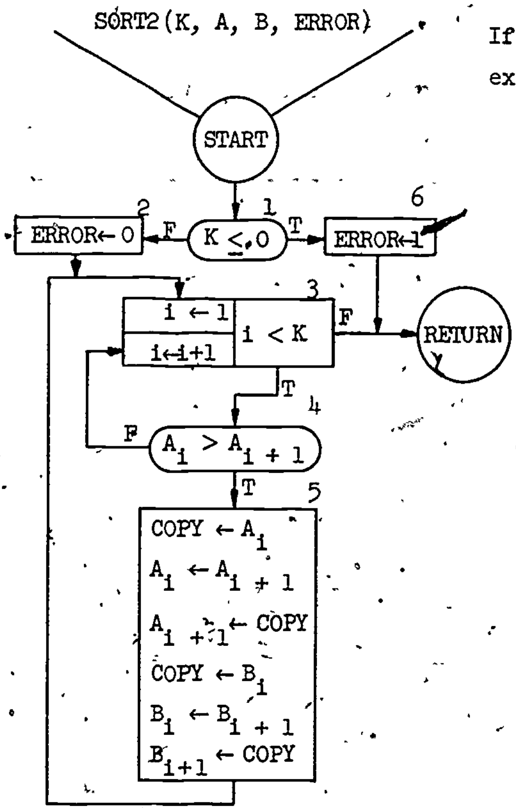
(d)


cxdiv(a1,b1,a2,b2,a,b)
START
denom ← a2 × a2 + b2 × b2
a ← (a1×a2 + b1×b2)/denom
b ← (a2×b1 - a1×b2)/denom
RETURN

2. (e)



3.



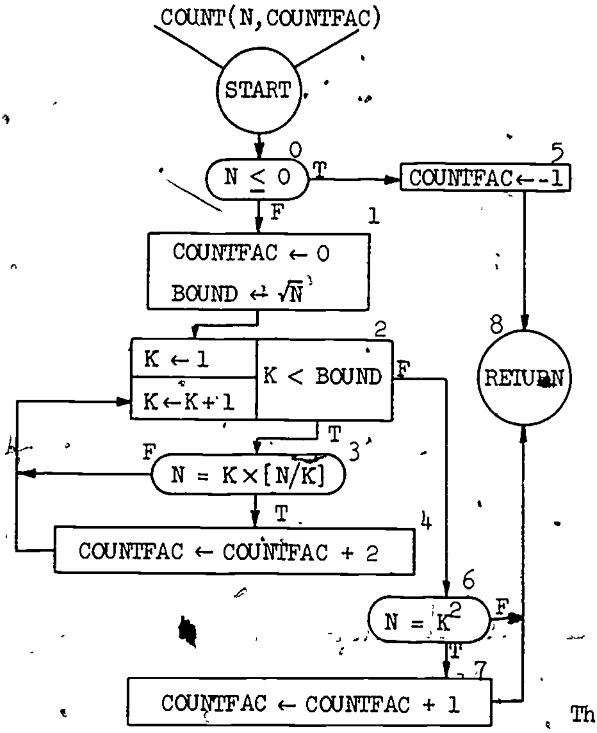
If ERROR = 1 then $K \leq 0$. A normal exit is indicated by ERROR = 0.

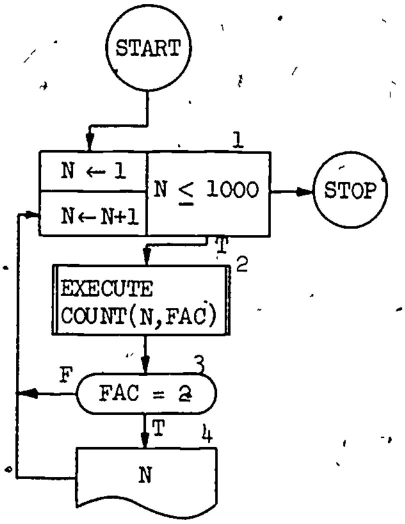
Comment: In this problem solution and in the three parts of Problem 5 in the same set we have used for brevity a different notation for treating vectors in the funnels of procedures. For the student you had best write:

SORT2(K, $\{A_i,\ i = 1(1)K\}$,
$\{B_i,\ i = 1(1)K\}$, ERROR).

Or, alternately, the student might use the more efficient shuttle-interchange method of Figure 4-34 to sort vector A.

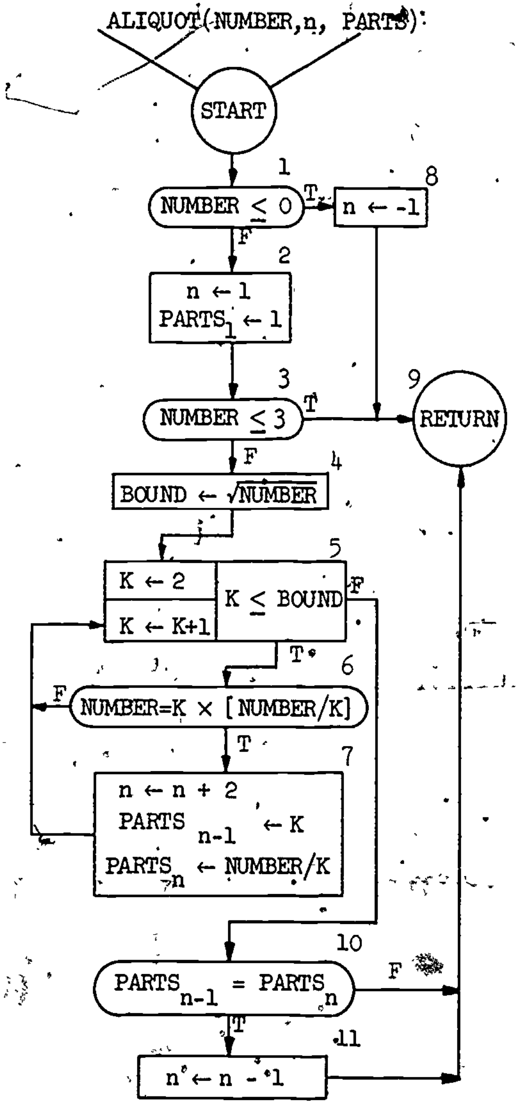
4. (a) A negative value of COUNTFAC (b) indicates $N \leq 0$.



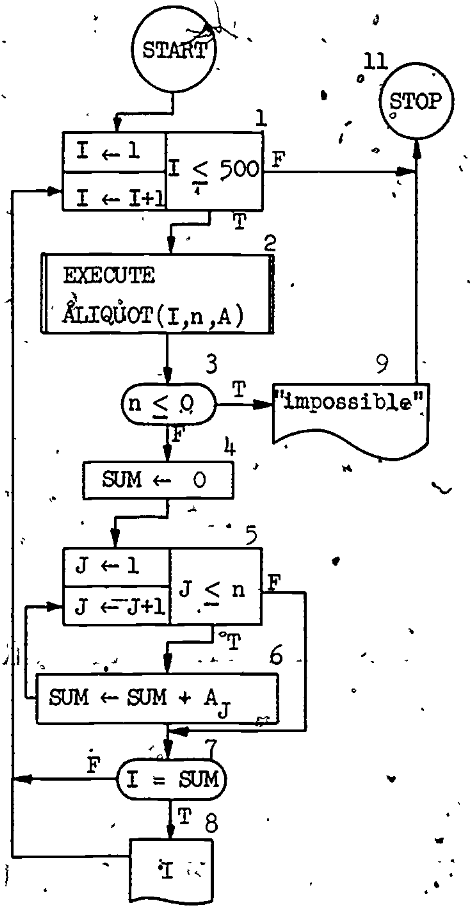


The algorithm is very inefficient because we must count all the factors of each number ≤ 1000, whereas as soon as we know that a number has more than two factors we know it is not a prime.

5. (a) Let $n \leftarrow -1$ indicate number ≤ 0.

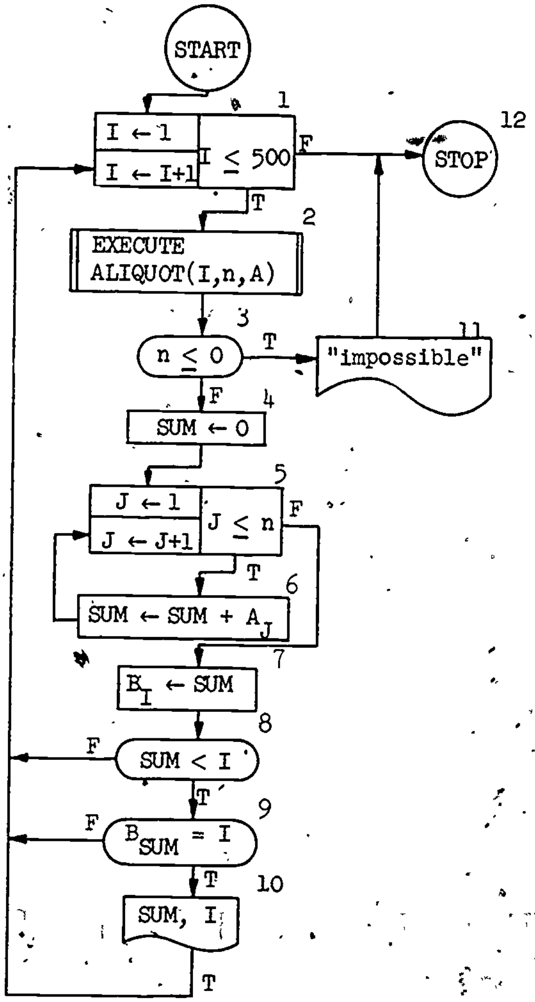


(b)



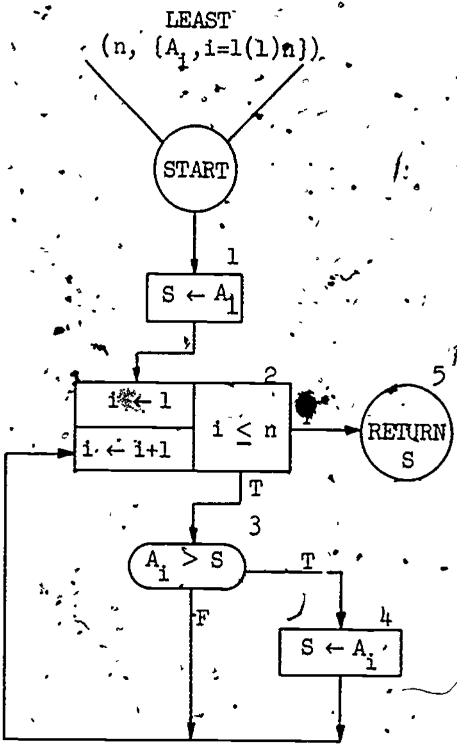
Remark: See comment on Problem 3.

5. (c) Let B be an array. B_i is the sum of the aliquot parts for the number i.

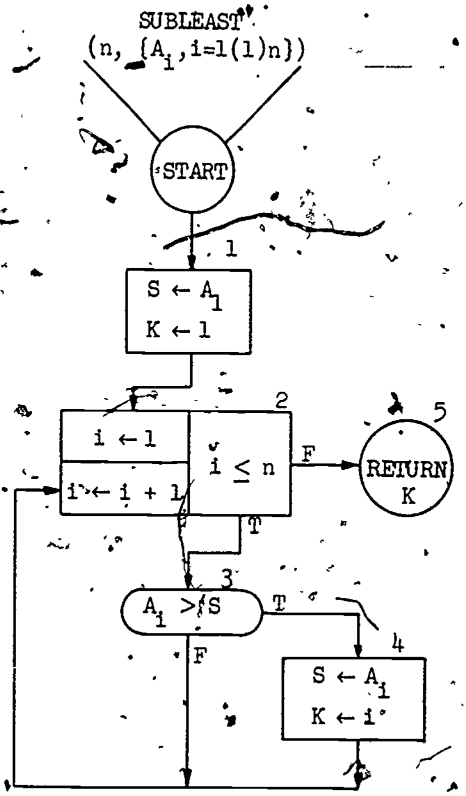


Answers to Exercises 5-4 - Set B

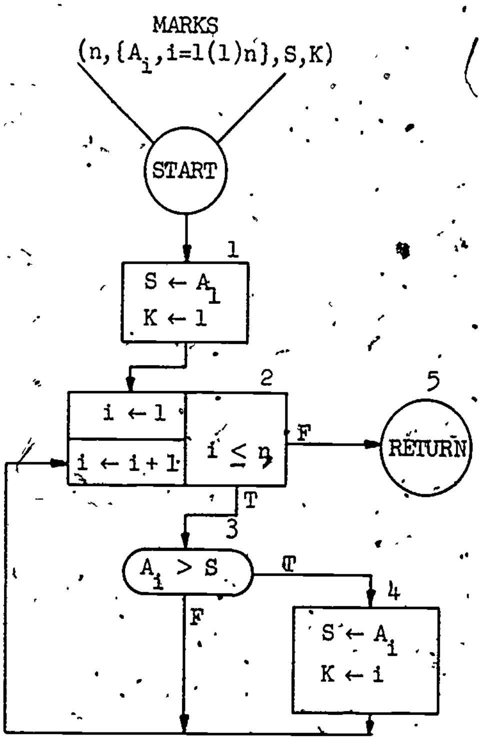
1.



2.



3.

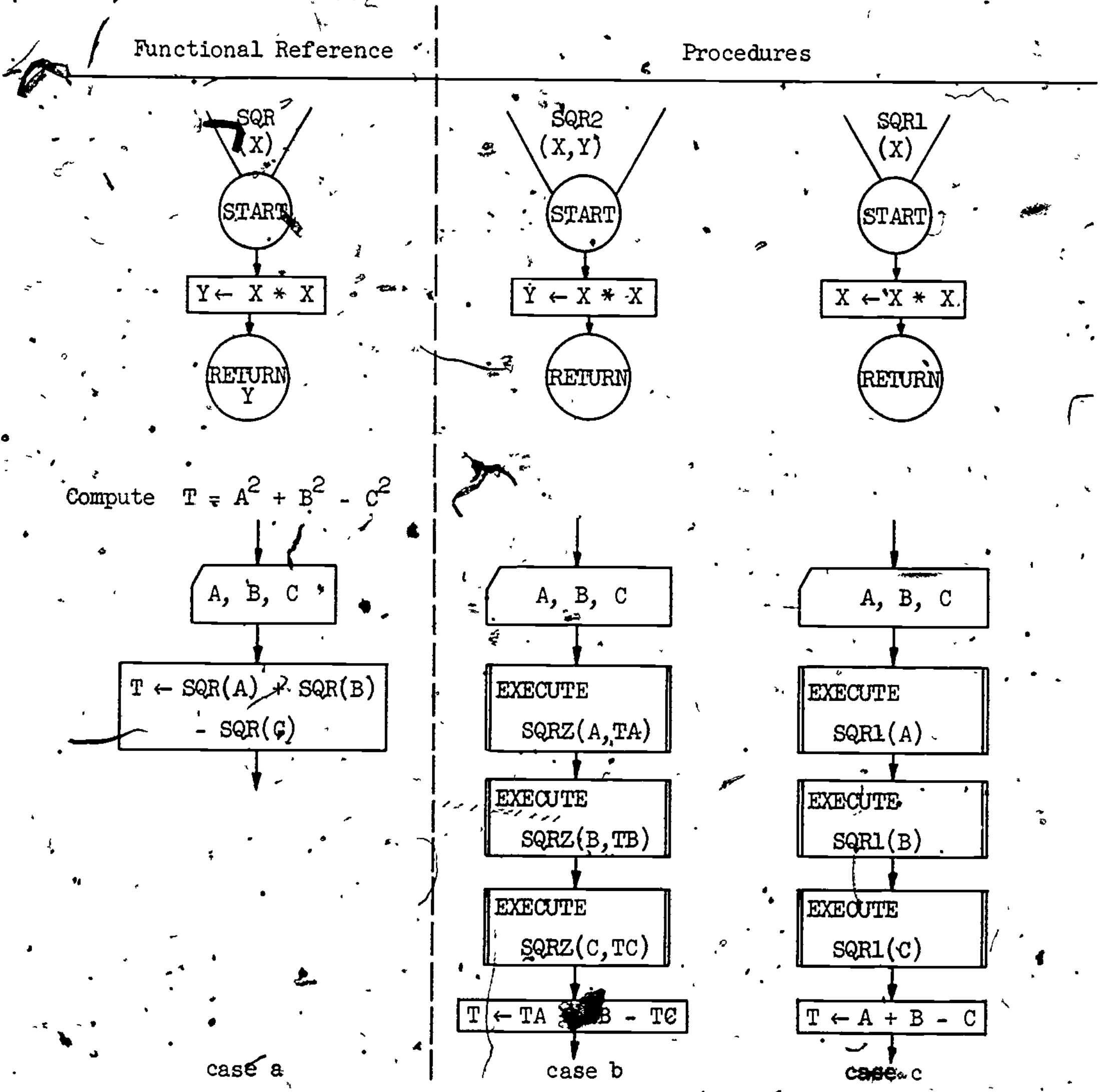


The effect of MARKS cannot be achieved with a functional reference because two values, S and K, must be determined by the procedure. A functional reference can only return 1 value.

Comment: The three exercises in Set B can be used to clarify a point that is often confusing. That is, when to use a function and when to use a procedure. The general answer is: If there is one value to be calculated, use a function; if more than one value has to be calculated, you must use a procedure.

Since procedures are more general (any function can be written as a procedure), why have two kinds of reference methods?. Why not always use procedures(Because functions, yielding just one value, can be used in arithmetic expressions but procedures cannot. The added generality of procedures means that programs to carry out procedures must take account of the most general case.

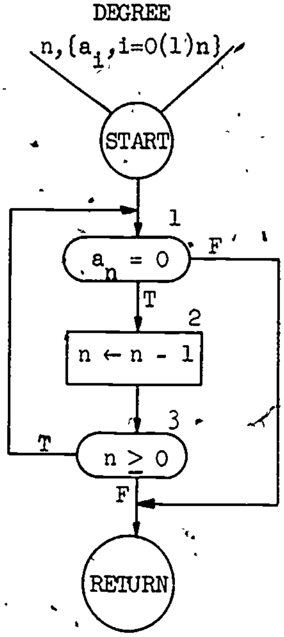
The following diagram for the simple operation of squaring can be helpful in comparing functional references and procedures in class discussion.



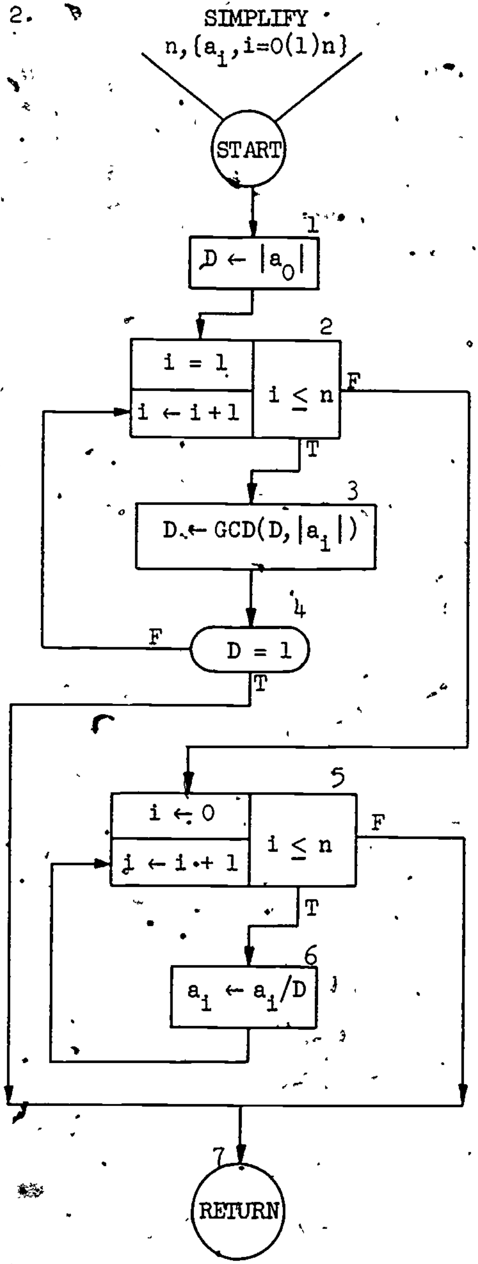
Comparing a functional reference with procedures which can accomplish a similar action.

Answers to Exercises 5-4 Set C

1.

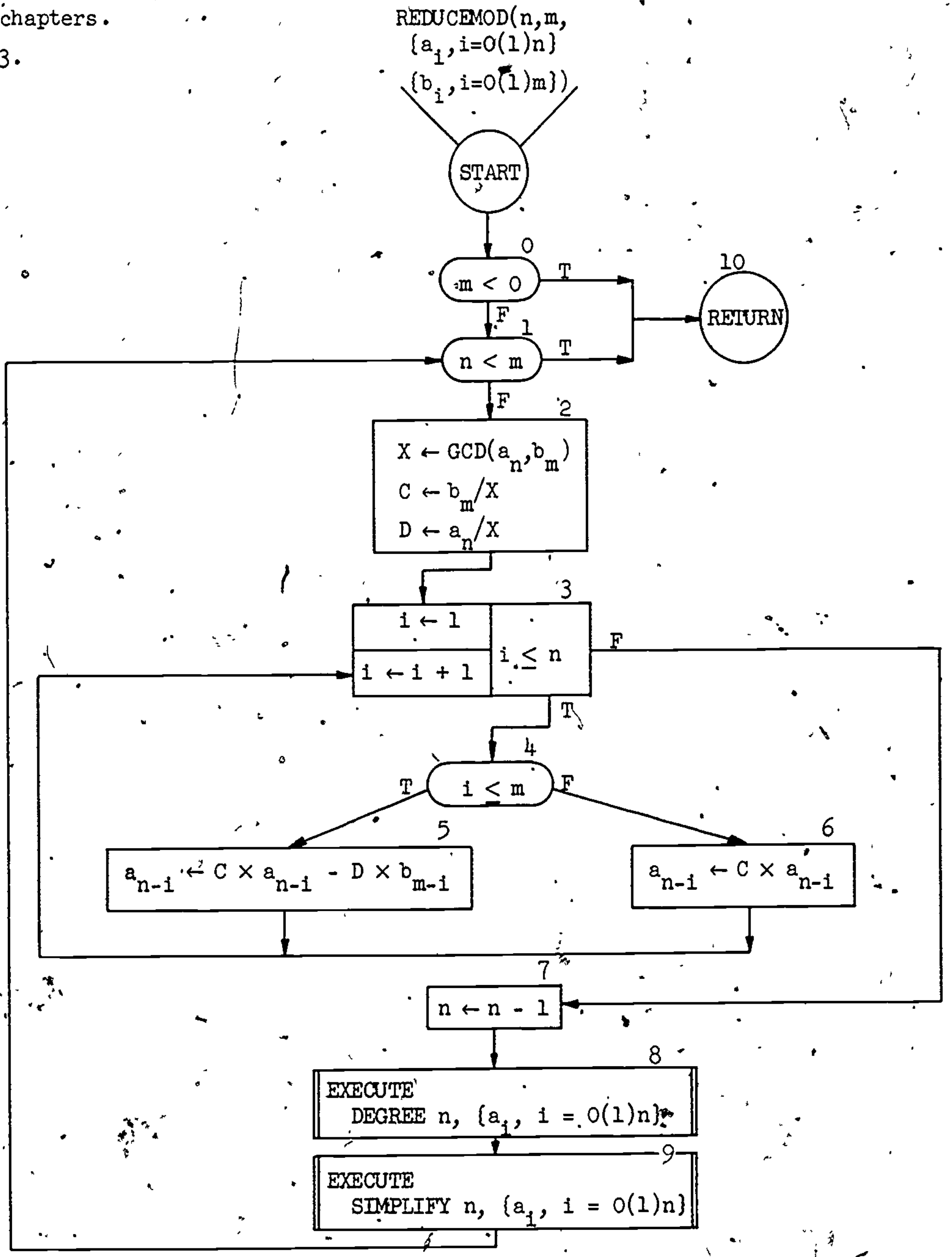


2.



Problems 3 and 4 of this section are very difficult. The teacher may wish to assign them as a project for the student to work on alone or in groups. The students may spend weeks on these problems. Hints could be given from time to time if the students get stuck. The problem material is rather sparse in Chapters 6 and 7 so that such a project would give the student something to work on during the period in which these chapters are taken up in class. Problem 4 uses problem 3 and both use problems 2 and 1. In addition, both problems 3 and 4 involve other flow charts developed in this and earlier chapters.

*3.



REDUCEMOD(n,m, $\{a_i, i=0(1)n\}$ $\{b_i, i=0(1)m\}$)
START
0
$m < 0$
T
F
10
RETURN
1
$n < m$
T
F
2
$X \leftarrow GCD(a_n, b_m)$
$C \leftarrow b_m/X$
$D \leftarrow a_n/X$
3
$i \leftarrow 1$
$i \leftarrow i + 1$
$i \leq n$
F
T
4
$i < m$
T
F
5
$a_{n-i} \leftarrow C \times a_{n-i} - D \times b_{m-i}$
6
$a_{n-i} \leftarrow C \times a_{n-i}$
7
$n \leftarrow n - 1$
8
EXECUTE DEGREE n, $\{a_i, i = 0(1)n\}$
9
EXECUTE SIMPLIFY n, $\{a_i, i = 0(1)n\}$

Discussion: REDUCEMOD has as its purpose the computation of the remainder when $a(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ is divided by $b(x) = b_m x^m + \ldots + b_1 x + b_0$. It is simply the division algorithm for polynomials.

The division theorem for polynomials tells us that for any two polynomials $a(x) = a_n x^n + \ldots + a_1 x + a_0$ and $b(x) = b_m x^m + \ldots + b_1 x + b_0$ (where not all coefficients of $b(x)$ are zero) there are uniquely determined polynomials $q(x)$ and $r(x)$ so that

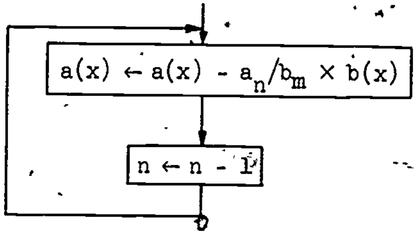
$$(1) \qquad a(x) = q(x) \cdot b(x) + r(x).$$

If degree $a(x) <$ degree $b(x)$, then $q(x) = 0$ and $r(x) = a(x)$.

The division algorithm enables us to compute $q(x)$ and $r(x)$ by an iterative process. We illustrate by letting $a(x) = 2x^4 + 2x^3 - 3x^2 + 7x + 2$ and $b(x) = 3x^2 - 5x + 1$. The first step in the division is shown below.

$$\begin{array}{r|l} & \frac{2}{3}x^2 \\ \hline 3x^2 - 5x + 1 & 2x^4 + 2x^3 - 3x^2 + 7x + 2 \\ & 2x^4 - \frac{10}{3}x^3 + \frac{2}{3}x^2 \\ \cline{2-2} & \qquad \frac{16}{3}x^3 - \frac{11}{3}x^2 + 7x + 2. \end{array}$$

Note that we have subtracted $\frac{2}{3}x^2 \cdot b(x)$ from $a(x)$. The next step amounts to repeating the process with the same $b(x)$ but with $a(x)$ replaced by $\frac{16}{3}x^3 - \frac{11}{3}x^2 + 7x + 2$. This observation will be reflected in our flow chart. We see that $\frac{2}{3}x^2$ has for its coefficient a_n/b_m and for its degree, $n - m$. We could conceive of a "generalized flow chart" component such as



The above process has one great drawback for computing purposes. The drawback lies in the appearance of fractions among the coefficients. The effect of this is that our answer will be annihilated by round-off. This difficulty can be handled by multiplying by the constant b_m, or better, by $b_m/\mathrm{GCD}(a_n, b_m)$, which is 3 in this case. Applied to the preceding example this would give:

$$\begin{array}{r|l} & 2x^2 \\ \hline 3x^2 - 5x + 1 & 6x^4 + 6x^3 - 9x^2 + 21x + 6 \\ & 6x^4 - 10x^3 + 2x^2 \\ \cline{2-2} & \qquad 16x^3 - 11x^2 + 21x + 6. \end{array}$$

The net effect of multiplying $a(x)$ by this integer constant is to multiply $a(x)$ and $r(x)$ by the same integer constant (see (1)).

As mentioned in the statement of the problem, multiplication of our polynomials by rational numbers will not affect divisibility properties.

Now we show the work without the powers of x being written down (i.e., in synthetic form).

$$\begin{array}{rrr|rrrrrl}
 & & & 2 & & & & & \\ \cline{4-8}
3 & -5 & 1 & 6 & 6 & -9 & 21 & 6 & \text{(i)} \\ \cline{1-3}
 & & & 6 & -10 & 2 & & & \text{(ii)} \\ \cline{4-8}
 & & & 0 & 16 & -11 & 21 & 6 & \text{(iii)}
\end{array}$$

If we let $C = b_m/ACD(a_n,b_m)$ and $D = a_n/ACD(a_n,b_m)$, then on line (i) we see the coefficients of $C \cdot a(x)$, on line (ii) the coefficients of $D \cdot b(x)$ and on (iii) the coefficients of $C \cdot a(x) - Dx^{n-m}b(x)$. In order, from left to right, these coefficients are

$$C \cdot a_{n-i} - D \cdot b_{m-i} \quad i = 0,1,\ldots,m$$

(2) followed by

$$C \cdot a_{n-i} \quad i = m+1,\ldots,n.$$

Now we fix our attention on the flow chart for problem 3.

In box 2 we compute the C and D described above. In boxes 3 through 6 we compute the coefficients of the new $a(x)$ as described in (2).

One trick should be noted to avoid confusion. Since the leading coefficients have been made equal by the choice of C and D, the leading coefficient of the new $a(x)$ will be zero. Knowing this we do not compute it in box 5 but, rather, leave a_n as it is. But then we reduce the value of n by 1 in box 9 so that the erroneous value is eliminated from further consideration. This technique is mildly interesting and can be used in other places and, in fact, is used in Chapter 7 in the section on solutions of systems of linear equations. The "trick" can be eliminated by initiating i at 0 in box 3 and dropping box 7 out of the flow chart.

In box 8 we compute the degree of the new $a(x)$ which may by some fluke have been decreased by more than 1 in the preceding steps. In box 9 we simplify $a(x)$ in the desperate hope of keeping the coefficients from growing too large and going off scale.

Returning to box 1 we ascertain whether the degree of $a(x)$ is now less than the degree of $b(x)$. If so, then we are through and $a(x)$ is our remainder as observed earlier. If not, we repeat the process.

In box 0, if $m = 0$, then $b(x)$ is a non-zero constant polynomial and we know in advance that the remainder will be zero so that there is no need to compute it. If $m < 0$, then $b(x)$ is identically zero and the division cannot be performed.

A final warning is in order concerning this problem and the next one. In order to avoid the occurrence of fractions and the consequent roundoff, we have called for repeated multiplication of our polynomials by suitable integers. The maximum number of times that this multiplication can occur in the execution of problem 4 is 1 less than the sum of the degrees of the two polynomials. The result is that although our program calls for very little storage space and relatively few arithmetic operations, on the other hand, it requires enormous word length. This is the price we have to pay for demanding exact answers. But, in this case, the alternative of rounding is not available to us. The result would be completely lost in roundoff.

To give some idea of the word length required, suppose that the degree of $a(x)$ is 8 ~~and~~ the degree of $b(x)$ is 7. Let M be the greatest of the absolute values of the coefficients of the two polynomials. Then the maximum of the magnitudes of the coefficients of the new $a(x)$ after the first pass through box 7 of the REDUCEMOD program cannot exceed

$$2M^2.$$

A new bound on the size of the coefficients on the successive passes through box 7 is obtained by squaring and then doubling the old bound. Then on the completion of the program of problem 4, if 14 passes through box 7 are required, our bound will be

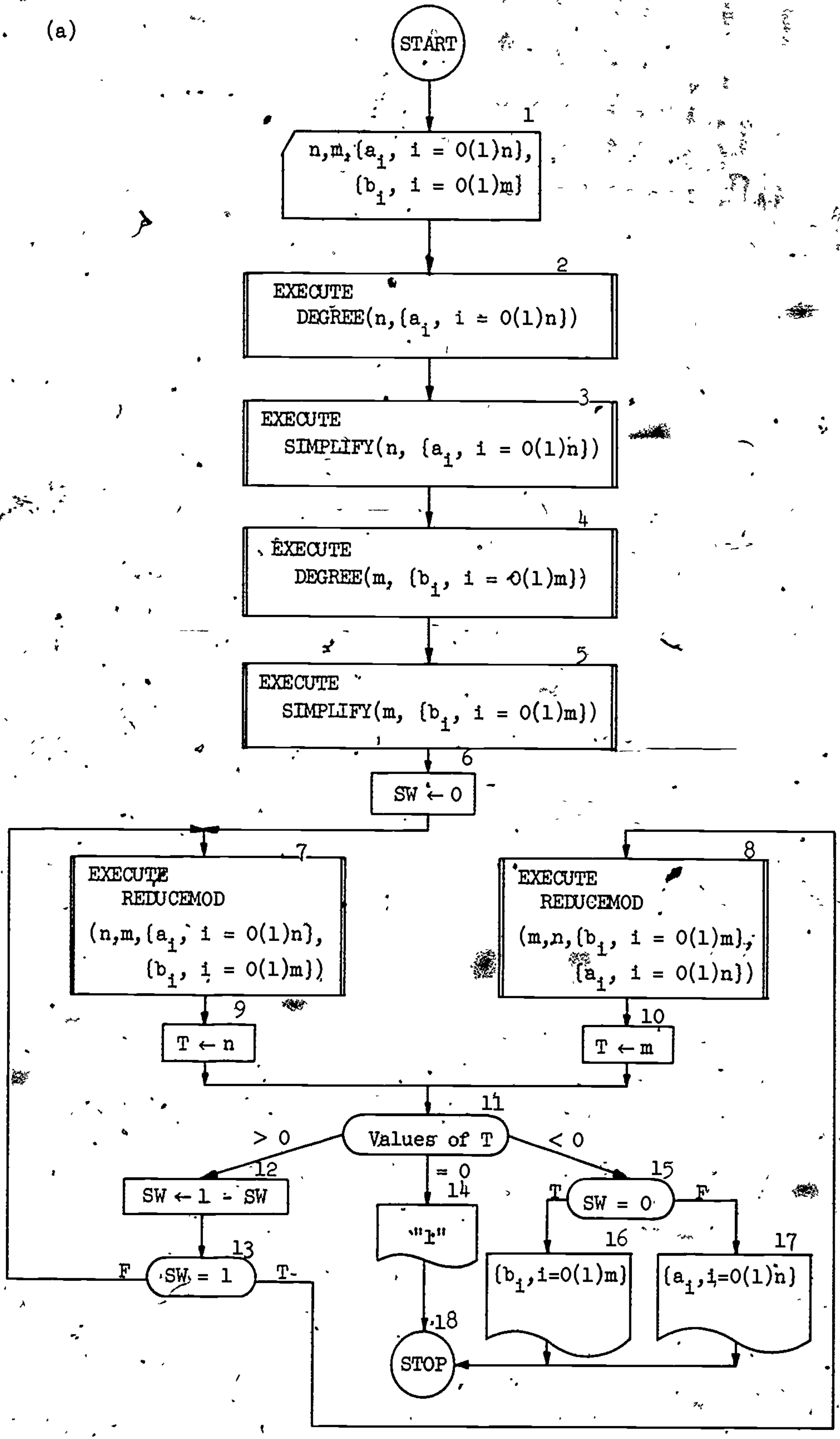
$$2\left(\cdots 2\left(2\left(2\left(2\left(2\left(2(2M^2)^2\right)^2\right)^2\right)^2\right)^2\right)^2 \cdots\right)^2$$

or

$$2^{2^{14}-1}M^{2^{14}}.$$

This is a ridiculously large number but hopefully, in practice, things will not turn out all that badly. Still, they are likely to be bad enough.

*4. (a)


START
1
n,m,$\{a_i$, i = 0(1)n},
$\{b_i$, i = 0(1)m}
2
EXECUTE
DEGREE(n,$\{a_i$, i = 0(1)n})
3
EXECUTE
SIMPLIFY(n, $\{a_i$, i = 0(1)n})
4
EXECUTE
DEGREE(m, $\{b_i$, i = 0(1)m})
5
EXECUTE
SIMPLIFY(m, $\{b_i$, i = 0(1)m})
6
SW ← 0
7
EXECUTE
REDUCEMOD
(n,m,$\{a_i$, i = 0(1)n},
$\{b_i$, i = 0(1)m})
8
EXECUTE
REDUCEMOD
(m,n,$\{b_i$, i = 0(1)m},
$\{a_i$, i = 0(1)n})
9
T ← n
10
T ← m
11
Values of T
> 0
= 0
< 0
12
SW ← 1 - SW
13
F
SW = 1
T
14
"1"
15
T
SW = 0
F
16
$\{b_i$,i=0(1)m}
17
$\{a_i$,i=0(1)n}
18
STOP

Discussion: The greatest common divisor algorithm for polynomials amounts to this: We start out with two polynomials $a^{(0)}(x)$ and $a^{(1)}(x)$. Then we ~~construct a sequence of polynomials~~

$$a^{(0)}(x), a^{(1)}(x), a^{(2)}(x), \ldots, a^{(h-1)}(x), a^{(h)}(x)$$

so that for $k > 1$, $a^{(k)}(x)$ is the remainder on dividing $a^{(k-2)}(x)$ by $a^{(k-1)}(x)$. When $a^{(h)}(x)$ is finally identically zero, we stop the process and $a^{(h-1)}(x)$ is the greatest common divisor of $a^{(0)}(x)$ and $a^{(1)}(x)$. The justification is the same as that for integers given in Section 3-2 of the student text. We have taken the liberty of multiplying our polynemials by constants at various stages in order to avoid fractions. This, as mentioned in the statement of the problem, will not alter our final answer.

We see that new polynomials in the above list will be generated by applying REDUCEMOD to the last two members of the list already computed. REDUCEMOD will call the last two members of the list $a(x)$ and $b(x)$, and will replace $a(x)$ by the remainder. Before repeating the process we must then switch the roles of $a(x)$ and $b(x)$.

This switching is accomplished in the main flow chart of problem 4 by the auxiliary switching variable SW seen in boxes 6, 12 and 13. This variable alternates between 0 and 1 on each execution of REDUCEMOD and thus sends us alternately through boxes 7 and 8. In these boxes we see the alternation of the roles of $a(x)$ and $b(x)$.

Boxes 9, 10 and 11 have the purpose of determining whether we are through. T is the degree of the remainder. If $T > 0$, then we are not through. If $T = 0$, then the remainder is a non-zero constant and we know that the greatest common divisor in simplest form will be 1, so we print out "1) (box 14). If $T < 0$ (that is, $T = -1$), then the remainder is zero and the last non-zero polynomial in our list is printed out as the greatest common divisor (boxes 15 - 17).

The following flow chart is a variant of the one just seen. The reduction from 17 boxes to 10 is achieved by the use of double subscripts. The two polynomials are

$$a_{0,n_0}x^{n_0} + a_{0,n_0-1}x^{n_0-1} + \ldots + a_{0,1}x + a_{0,0}$$

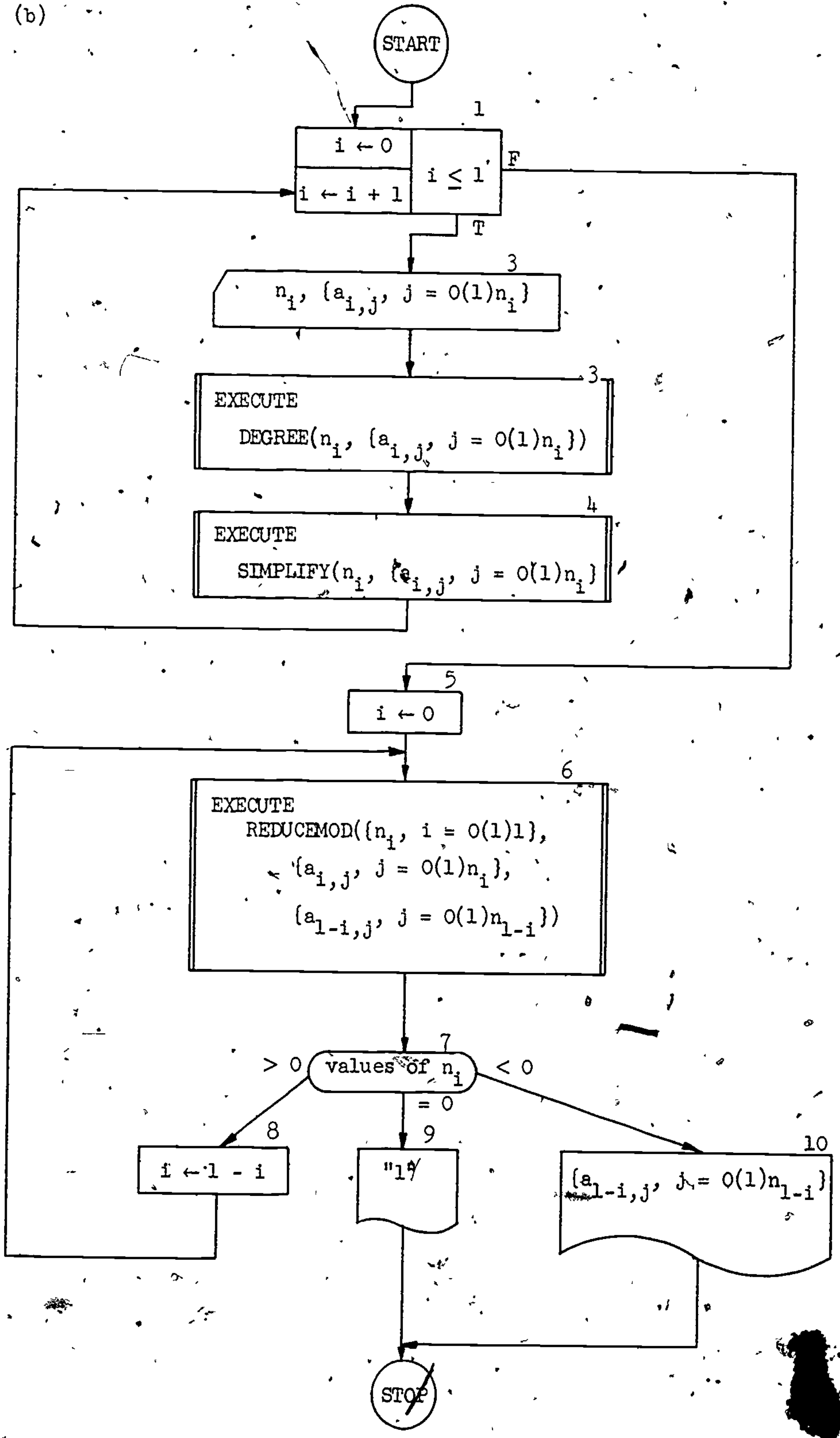
and

$$a_{1,n_1}x^{n_1} + a_{1,n_1-1}x^{n_1-1} + \ldots + a_{1,1}x + a_{1,0} .$$

The switching variable SW now becomes i so that the switching is accomplished by alternating the values of the first subscript, thus changing the roles of the two polynomials. Now boxes 2 through 5 of 4(a) are compressed into boxes 3 and 4 of 4(b). Boxes 7 and 8 of 4(a) are compressed into box 6 of 4(b). Boxes 9 through 11 of 4(a) are compressed into box 7 of 4(b). Boxes 12 and 13 of 4(a) are compressed into 8 of 4(b). Boxes 15 through 17 of 4(a) are compressed into box 10 of 4(b). Boxes 6 and 14 of 4(a) become, respectively, boxes 5 and 9 of 4(b).

T5

*4. (b)


START
1
$i \leftarrow 0$
$i \leftarrow i + 1$
$i \leq l$
F
T
3
$n_i, \{a_{i,j}, j = 0(1)n_i\}$
3
EXECUTE
$\text{DEGREE}(n_i, \{a_{i,j}, j = 0(1)n_i\})$
4
EXECUTE
$\text{SIMPLIFY}(n_i, \{a_{i,j}, j = 0(1)n_i\}$
5
$i \leftarrow 0$
6
EXECUTE
$\text{REDUCEMOD}(\{n_i, i = 0(1)l\},$
$\{a_{i,j}, j = 0(1)n_i\},$
$\{a_{l-i,j}, j = 0(1)n_{l-i}\})$
7
> 0
values of n_i
< 0
$= 0$
8
$i \leftarrow l - i$
9
"1"
10
$\{a_{l-i,j}, j = 0(1)n_{l-i}\}$
STOP

Further work could be done along the lines of this problem set: The flow chart of problem 4 could be made into a procedure which we might call GRCOMDIV (to distinguish it from GCD). Now we could, given a polynomial $a(x)$ with integer coefficients, find a polynomial SIMP $a(x)$ having the same roots as $a(x)$ but having no multiple roots. This polynomial is given by

$$\text{SIMP } a(x) = a(x)/b(x)$$

where

$$b(x) = \text{GRCOMDIV}(a(x), a'(x)),$$

$a'(x)$ being the derivative of $a(x)$.

Next, one could take up Sturm's Method for isolating the real roots of polynomials. (See, for example, Uspensky, Theory of Equations, McGraw-Hill, New York, 1948, Chapter VII.)

For illustrative purposes, we show in the following table the repeated trace through REDUCEMOD for the solution of the GCD of

$$a(x) = 3x^5 + ax^4 + 2x^3 - x^2 + 5x + 2$$

and

$$b(x) = 2x^4 + 6x^3 + 3x^2 + 3x + 1 .$$

The GRCOMDIV($a(x)$, $b(x)$) turns out to be $x^2 + 3x + 1$.

As can be seen on the trace, three successive calls must be made on REDUCEMOD before a remainder is found which is identically zero (i.e., $n < 0$).

Box Number	n	m	X	C	D	i	a_5	a_4	a_3	a_2	a_1	a_0	b_4	b_3	b_2	b_1	b_0	True/ False
First entry to REDUCEMOD. (Initial values)	5	4					3	9	2	-1	5	2	2	6	3	3	1	
1																		F
2			1	2	3													
3						1												
4																		T
5								0										
3						2												
4																		T
5									-5									
3						3												
4																		T
5										-11								
3						4												
4																		T
5											7							
3						5												
4																		F
6												4						
3						6												
7	4																	
8	3								-5	-11	7	4	2	6	3	3	1	
9																		
1	4	3						2	6	3	3	1		-5	-11	7	4	(Exit) T
Second entry to REDUCEMOD																		
1																		F
2			1	-5	2													
3						1												
4																		T
5									-8									
3						2												
4																		T
5										-29								
3						3												
4																		T
5											-23							
3						4												
4																		F
5												-5						
3						5												
7	3	3																
8									-9	-29	-23	-5		-5	-11	7	4	
9																		
1																		

Box Number	n	m	X	C	D	i	a_5	a_4	a_3	a_2	a_1	a_0	b_4	b_3	b_2	b_1	b_0	True/ False
balance brought forward	3	3	1	-5	2	5			-8	-29	-23	-5		-5	-11	7	4	
2			1	-5	-8													
3						1												
4																		T
5										57								
3						2												
4																		T
5											171							
3						3												
4																		T
5												57						
3						4												
7	2																	
8	2									57	171	57		-5	-11	7	4	
9										1	3	1						
1																		(Exit) T
Third entry to REDUCEMOD	3	2							-5	-11	7	4			1	3	1	
1																		
2			1	1	-5													
3						1												
4																		T
5										4								
3						2												
4																		T
5											12							
3						3												
4																		F
5												4						
3						4												
7	2																	
8										4	12	4						
9										1	3	1						
1																		F
2			1	1	1													
3						1												
4																		T
5											0							
3						2												
4																		[illegible]
5												0						
3						3												
7	1																	
8	-1										0	0						
9																		
1														G.C.D.	1	3	1	(Exit) T

Supplementary exercises

(We reprint here a set of exercises which may be of interest to your students. These originally appeared in the student text of the preliminary edition. The solution set is given at the end of the reprint.)

Students often enjoy number-conversion problems. Problems 1, 3, 4 and 5 are reasonable for class assignment. Problems 2 and 6 are tricky and really better suited for challenging projects than for daily assignments.

(Reprint begins here)

The decimal system has 10 as a base; i.e., there are ten unique digits (symbols), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. With these ten digits we can write a number as large or as small as we wish by allowing the position of the digit to represent a specific power of the base. As you know, the decimal numeral 4047.4_{TEN} means

$$4047.4_{TEN} = (4 \times 10^3) + (0 \times 10^2) + (4 \times 10^1) + (7 \times 10^0) + (4 \times 10^{-1}).$$

Note the use of the subscript TEN to indicate the base.

When numbers are expressed in terms of bases other than ten we can easily find their decimal representations. For example, the binary (base two with symbols 0, 1) numeral 1101_{TWO} may be expressed

$$(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$

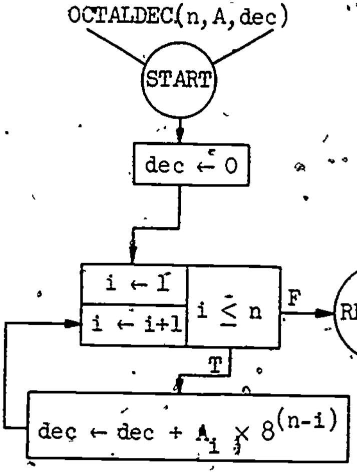
or 13_{TEN}. The octal (base eight symbols 0, 1, 2, 3, 4, 5, 6, 7) numeral 263_{EIGHT} becomes

$$(2 \times 8^2) + (6 \times 8^1) + (3 \times 8^0)$$

or 179_{TEN}. For the hexodecimal system (base sixteen, symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, U, V, W, X, Y, Z) $15W_{SIXTEEN}$ becomes

$$(1 \times 16^2) + (5 \times 16^1) + (12 \times 16^0) = 348_{TEN}.$$

Now let's look at a reference flow chart for procedure octaldec(n,A,dec) for the conversion of octal numerals into decimal, as given below. We let A be a vector of n places having as its entries the digits of a positive numeral arranged in the order in which they appear in the numeral; e.g., for 624, $n \leftarrow 3$, $A_1 \leftarrow 6$, $A_2 \leftarrow 2$, $A_3 \leftarrow 4$. The variable dec returns the decimal value.



Flow chart for conversion of an octal numeral into decimal

Wasn't it simple? Now it's your turn.

1. Adopting the flow chart of octaldec, prepare a reference flow chart for procedure intodec(n,A,B,dec) which converts a positive numeral, base b, into decimal. Base b will be restricted to $b < 10$.

2. For the ingenious student we propose the problem of converting from hexadecimal to decimal. In this case the entries of vector A will be alphanumeric, rather than numerical as before. Thus, we must begin by identifying each symbol.

   (a) Prepare a reference flow chart for a procedure identify(k,A,VALUE) which accepts the vector A and the index k of the element to be identified. It uses as local variables the alphanumeric elements of the vector COMPARE.

Vector COMPARE

Subscript	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Element	"0"	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"	"U"	"V"	"W"	"X"	"Y"	"Z"

The procedure returns VALUE, the value of the hexadecimal digit represented in A_i.

(b) Now prepare the reference flow chart for a procedure hexdecdec(n,A) which converts hexadecimal numbers into decimal numbers.

Now for the reverse process. To convert a decimal integer d to another base b, we divide d by b, obtaining a quotient q and a remainder which we store in the remainder vector r as r_1. We replace d by q and divide the new d by b. Again q becomes d; the remainder is stored as r_2. The process is repeated until q is 0. The digits of the answer are contained in vector r in reverse order, low-order to high-order.

The method may be illustrated in the conversion of 204_{TEN} into binary:

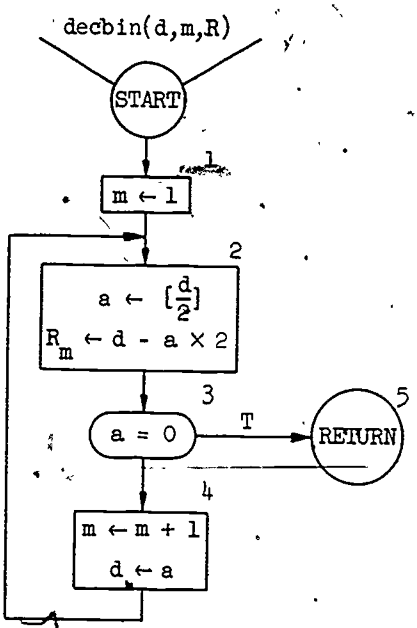
$$q = [\tfrac{204}{2}] = 102 \qquad r_1 = 0$$
$$q = [\tfrac{102}{2}] = 51 \qquad r_2 = 0$$
$$q = [\tfrac{51}{2}] = 25 \qquad r_3 = 1$$
$$q = [\tfrac{25}{2}] = 12 \qquad r_4 = 1$$
$$q = [\tfrac{12}{2}] = 6 \qquad r_5 = 0$$
$$q = [\tfrac{6}{2}] = 3 \qquad r_6 = 0$$
$$q = [\tfrac{3}{2}] = 1 \qquad r_7 = 1$$
$$q = [\tfrac{1}{2}] = 0 \qquad r_8 = 1$$

Thus, $204_{TEN} = 11001100_{TWO}$.

3. (a) Use the method to convert 1284_{TEN} into octal.

(b) Convert 6345_{TEN} into hexadecimal.

Now that you've practiced the method, we present the reference flow chart for procedure decbin(d,m,R) which converts the positive decimal numeral d into binary. The result is stored in the first m digits of vector R with the digits arranged in reverse order to that in which they appear in the numeral (i.e., 725 would appear as 527).



Flow chart for conversion of a decimal numeral into binary

Your turn again.

4. Write a reference flow chart for the procedure outdec(d,b,m,R) which converts the positive decimal numeral d to base b. Again the result is stored in the first m digits of vector R with the digits arranged in reverse order.

5. Now it's time to put the two parts together. Draw a flow chart which inputs the quantities bl, b2, n, A where the digits of a positive numeral in base bl are contained in the first n elements of vector A in normal order and where b2 is the base to which the numeral is to be converted. The output will be the numeral in base b2. We make the restriction that $bl \leq 10$ and $b2 \leq 10$.

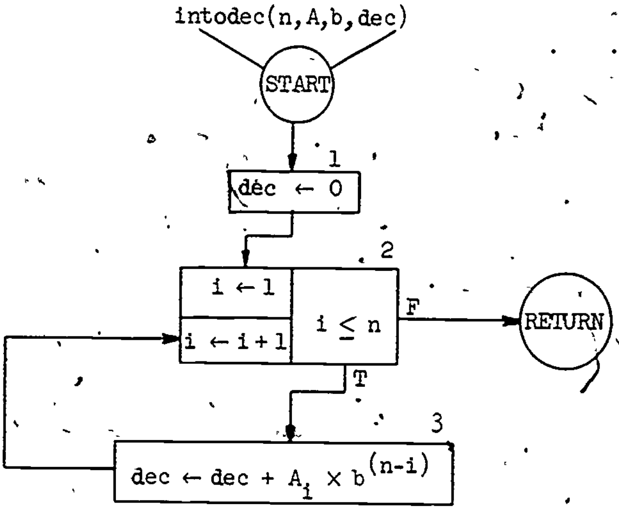
6. Finally, for the really crafty student we suggest a problem using Roman numerals as input. Prepare a reference flow chart for a procedure Rnum(n,A,NUM) which converts any Roman numeral less than MMM represented as a vector A of n elements into an Arabic numeral NUM. The vector A contains as elements each digit of the Roman numeral arranged high-order to low-order. As local variables use the elements of vectors ROMAN and VALUE. The comparison vector ROMAN contains the seven elements listed below. Each element of the vector VALUE contains the value of the corresponding element in ROMAN as given below.

Vector ROMAN	I	V	X	L	C	D	M
Vector VALUE	1	5	10	50	100	500	1000

(b) Draw the flow chart for a program which will input two Roman numerals less than MMM and output an echo check and their sum in Arabic numerals. Go back to the input step.

Solutions for preceding exercises

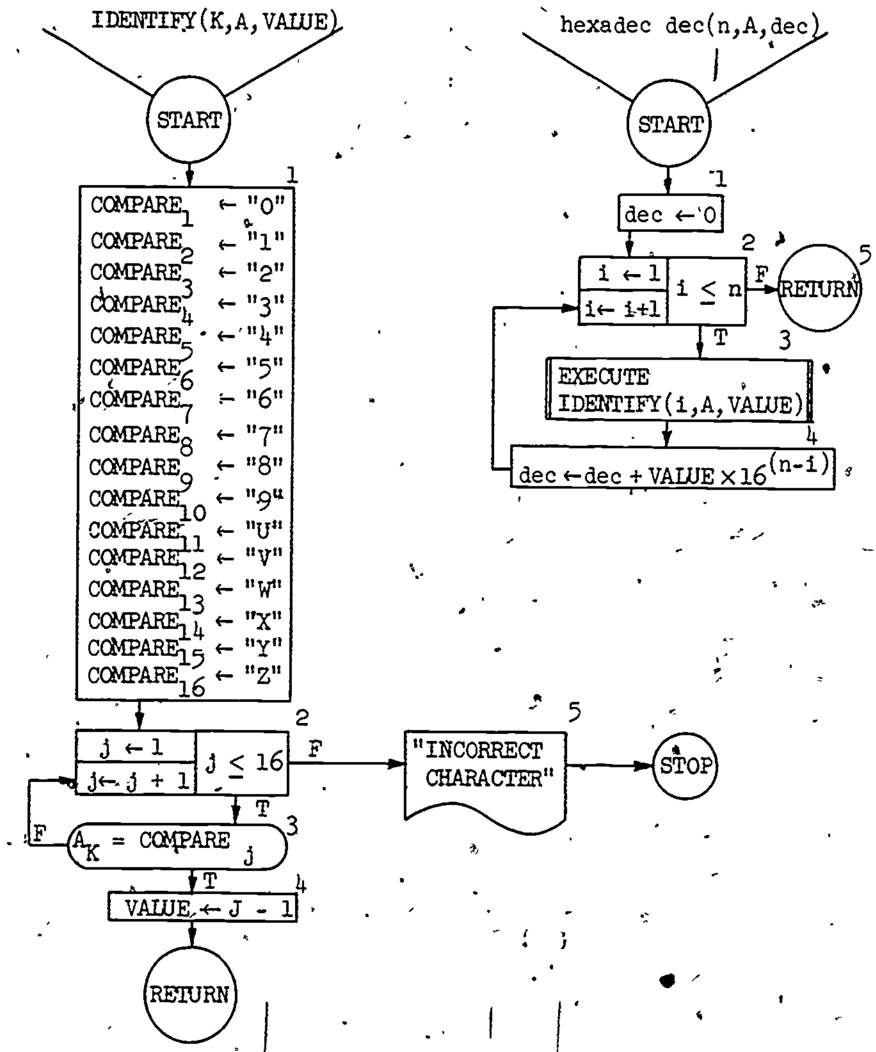
1.



For test data we suggest that the following be transformed to base ten:

Test Data	Answer
100101_{TWO}	37_{TEN}
436_{EIGHT}	286_{TEN}
122102_{THREE}	470_{TEN}
110100011_{TWO}	419_{TEN}
43701_{EIGHT}	17601_{TEN}
42153_{SIX}	5685_{TEN}

2. (a)



Suggested Test Data	Answer
$3XV_{SIXTEEN}$	987_{TEN}
$2Z4U_{SIXTEEN}$	12298_{TEN}

3. (a) $q = [\frac{1284}{8}] = 160 \quad r_1 = 4$ (b) $q = [\frac{6345}{16}] = 396 \quad r_1 = 9$

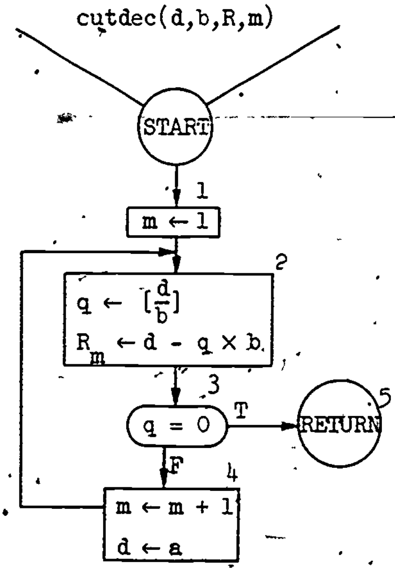
$q = [\frac{160}{8}] = 20 \quad r_2 = 0$ $q = [\frac{396}{16}] = 24 \quad r_2 = 12$

$q = [\frac{20}{8}] = 2 \quad r_3 = 4$ $q = [\frac{24}{16}] = 1 \quad r_3 = 8$

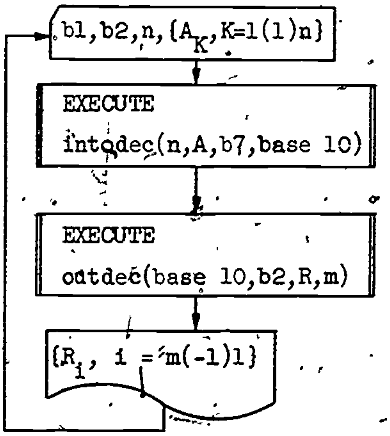
$q = [\frac{2}{8}] = 0 \quad r_4 = 2$ $q = [\frac{1}{16}] = 0 \quad r_4 = 1$

$1284_{TEN} = 2404_{EIGHT}$ $6345_{TEN} = 18W9_{SIXTEEN}$

4.

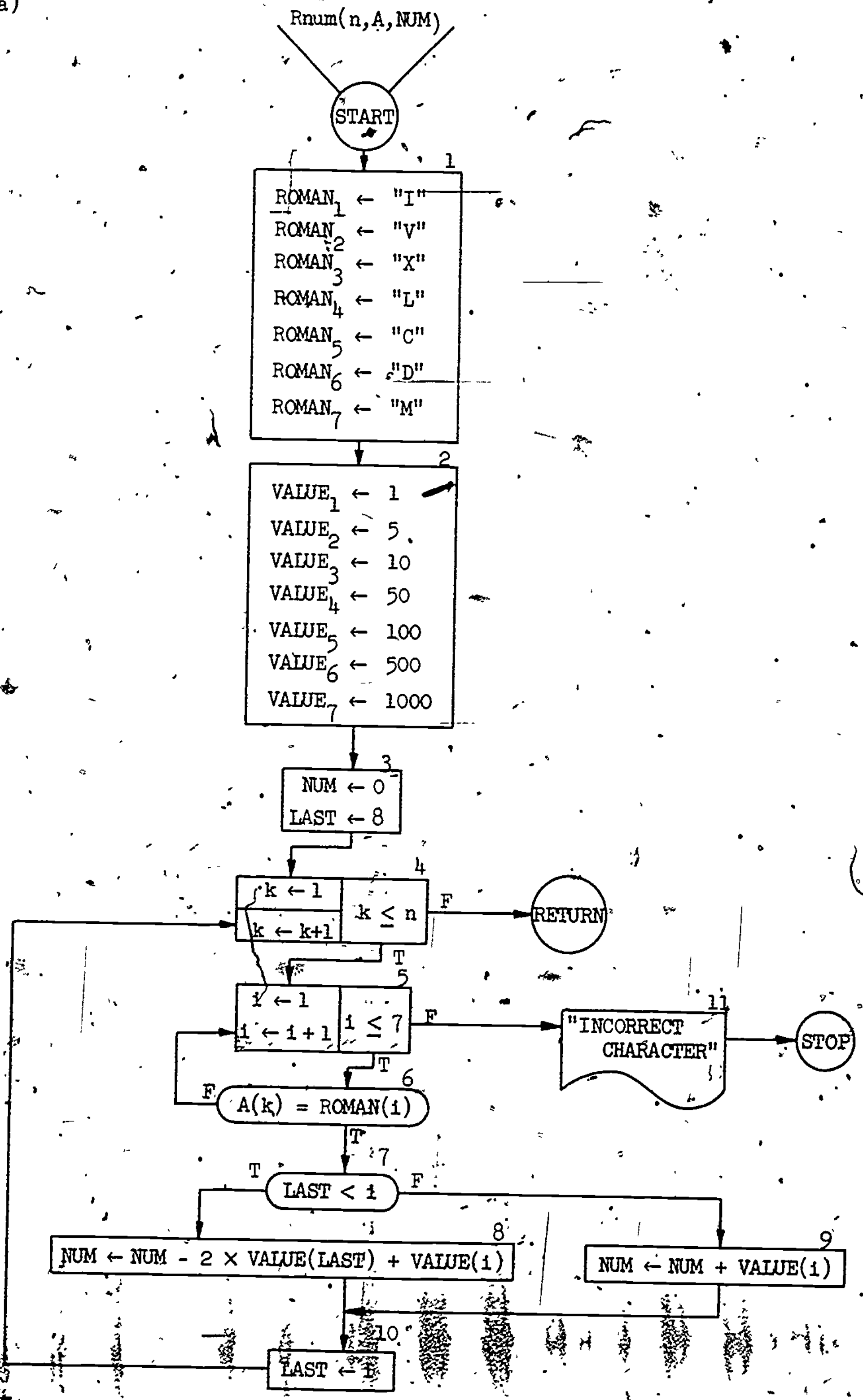


5.

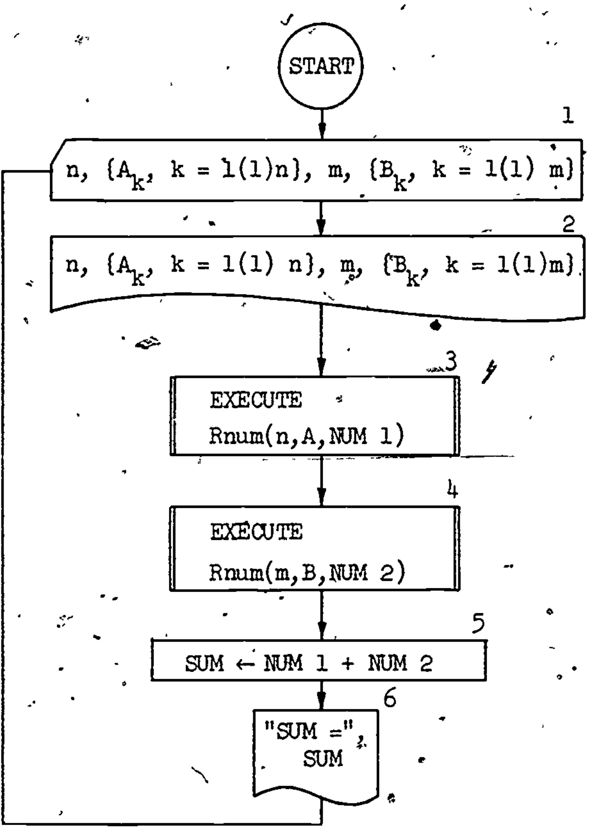


	Suggested Test Data		Answer
1.	13674_{EIGHT}	into base TWO	1011110111100_{TWO}
2.	6429_{TEN}	into base FIVE	201204_{FIVE}
3.	102112_{THREE}	into base EIGHT	467_{EIGHT}

6. (a)


Rnum(n,A,NUM)
START
1
ROMAN$_1$ ← "I"
ROMAN$_2$ ← "V"
ROMAN$_3$ ← "X"
ROMAN$_4$ ← "L"
ROMAN$_5$ ← "C"
ROMAN$_6$ ← "D"
ROMAN$_7$ ← "M"
2
VALUE$_1$ ← 1
VALUE$_2$ ← 5
VALUE$_3$ ← 10
VALUE$_4$ ← 50
VALUE$_5$ ← 100
VALUE$_6$ ← 500
VALUE$_7$ ← 1000
3
NUM ← 0
LAST ← 8
4
k ← 1
k ← k+1
k ≤ n
F
RETURN
T
5
i ← 1
i ← i + 1
i ≤ 7
F
11
"INCORRECT CHARACTER"
STOP
T
6
F
A(k) = ROMAN(i)
T
7
T
LAST < i
F
8
NUM ← NUM - 2 × VALUE(LAST) + VALUE(i)
9
NUM ← NUM + VALUE(i)
10
LAST ← i

6. (b)


START
1
n, {A_k, k = 1(1)n}, m, {B_k, k = 1(1) m}
2
n, {A_k, k = 1(1) n}, m, {B_k, k = 1(1)m}
3
EXECUTE
Rnum(n,A,NUM 1)
4
EXECUTE
Rnum(m,B,NUM 2)
5
SUM ← NUM 1 + NUM 2
6
"SUM =",
SUM

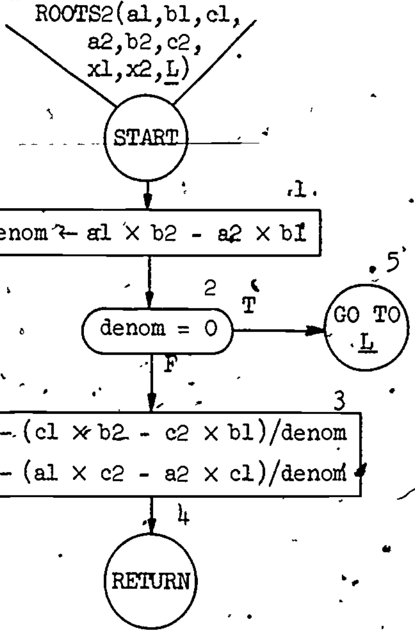
Suggested Test Data	Answer
1. MCDXIV, MMIX	SUM = 3423
2. LX, DXLV	SUM = 605
3. MMCVIII, CDLXI	SUM = 2569

Answers to Exercises 5-5

1. Assume equations of the form

$$a1 \times x + b1 \times y = c1$$
$$a2 \times x + b2 \times y = c2$$

Let the value of EXIT be zero if lines intersect, and one if the lines are parallel.


ROOTS2(a1,b1,c1,
a2,b2,c2,
x1,x2,L)
START
1
denom ← a1 × b2 - a2 × b1
2
denom = 0
T
F
5
GO TO
L
3
x1 ← (c1 × b2 - c2 × b1)/denom
x2 ← (a1 × c2 - a2 × c1)/denom
4
RETURN

Suggested Problems:

(a) $3x_1 + 2x_2 = -1$
$2x_1 - 5x_2 = 3$

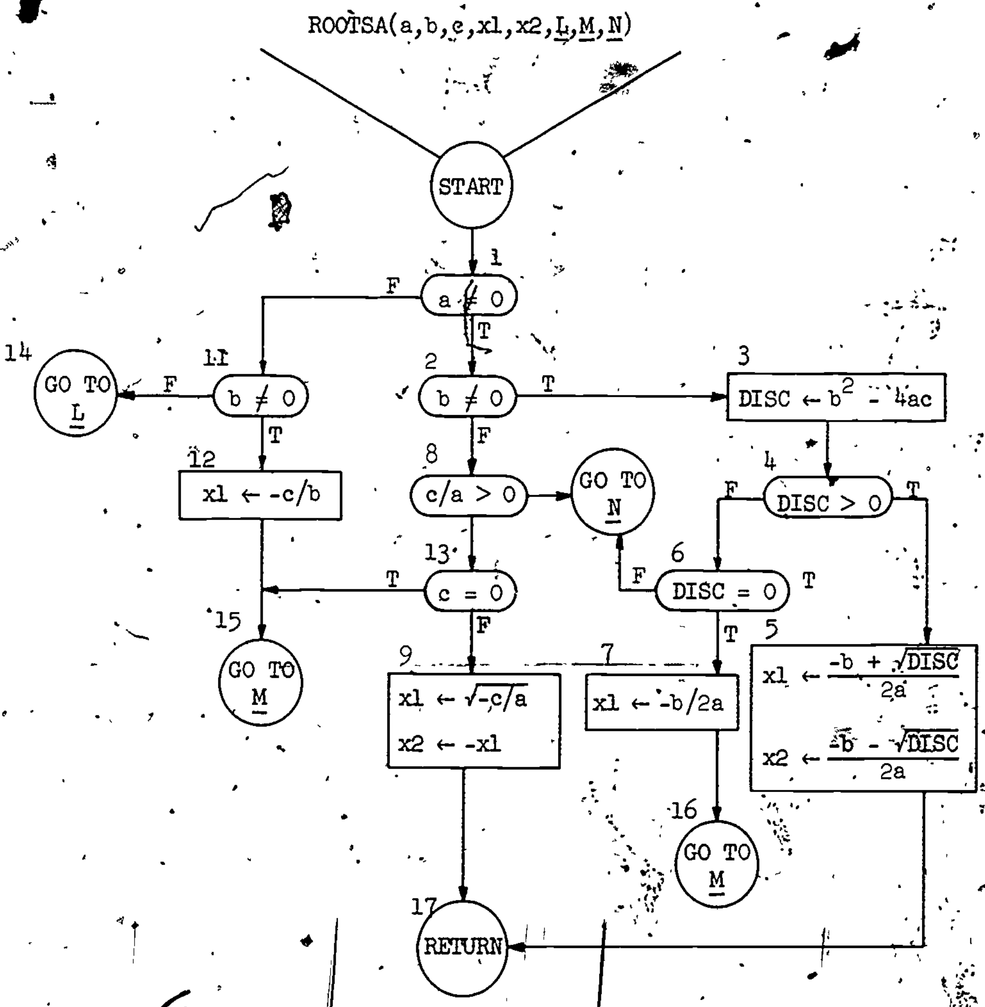
(b) $6.147x_1 - 3.28_2 = 2.481$
$3.2x_1 + 4.91x_2 = -1.233$

Answers:

(a) $x_1 = \frac{1}{19} = 0.0526$
$x_2 = -\frac{11}{19} = -0.579$

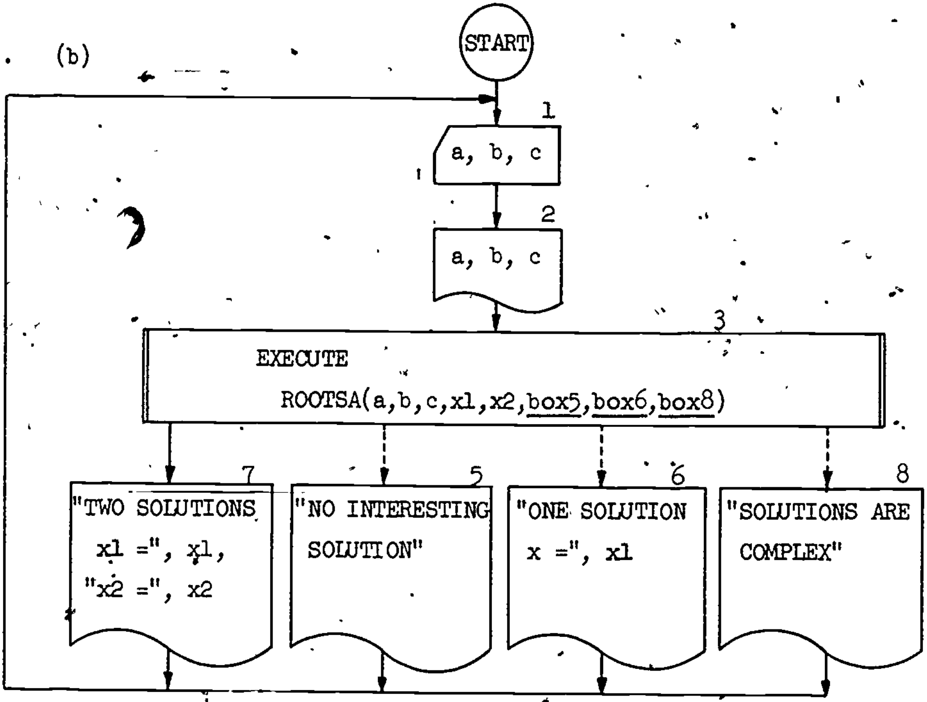
(b) $x_1 = 0.201$
$x_2 = -0.380$

2. (a)

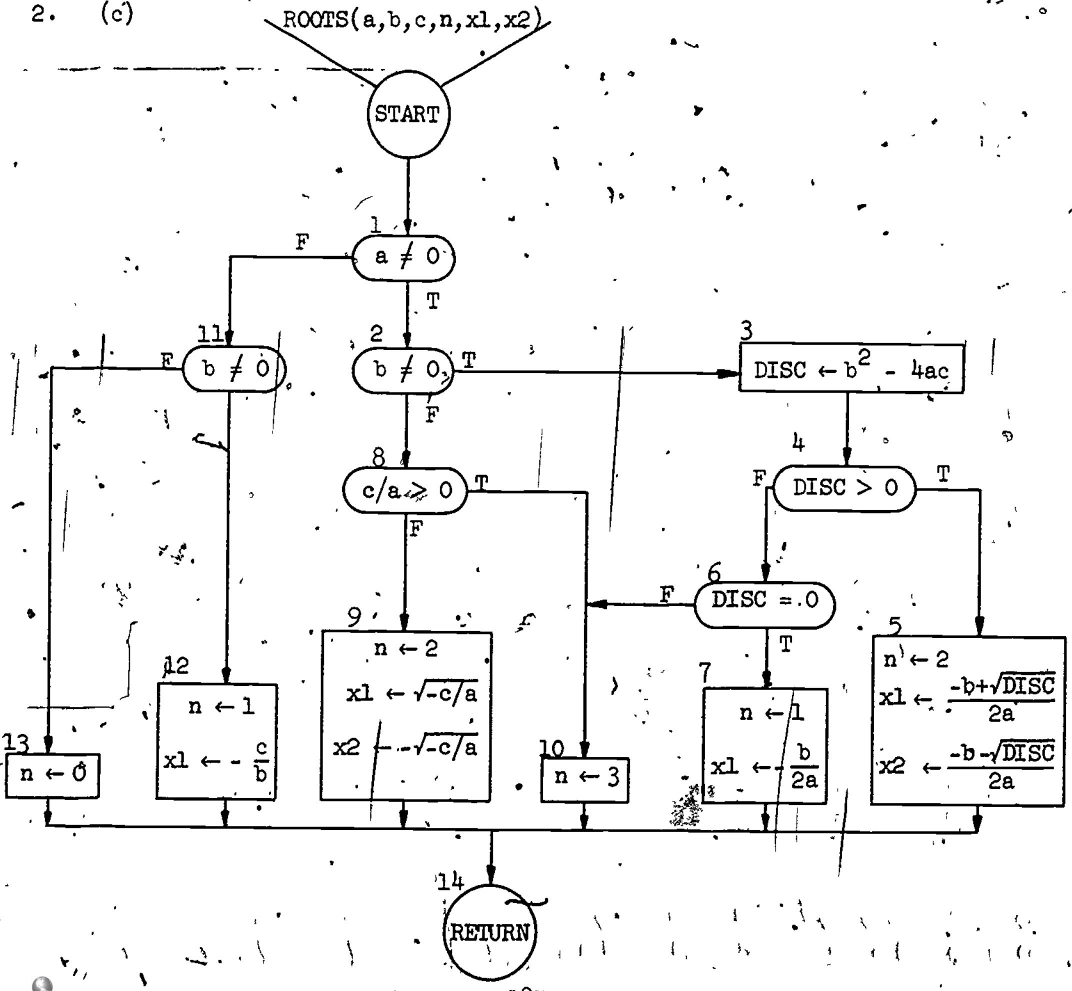

ROOTSA(a,b,c,x1,x2,L,M,N)
START
1
a ≠ 0
F
T
2
b ≠ 0
T
3
DISC ← b² - 4ac
4
DISC > 0
F
T
14
GO TO L
F
11
b ≠ 0
T
12
x1 ← -c/b
8
c/a > 0
GO TO N
6
DISC = 0
F
T
13
c = 0
T
F
15
GO TO M
9
x1 ← √-c/a
x2 ← -x1
7
x1 ← -b/2a
T
5
x1 ← (-b + √DISC)/2a
x2 ← (-b - √DISC)/2a
16
GO TO M
17
RETURN

T5

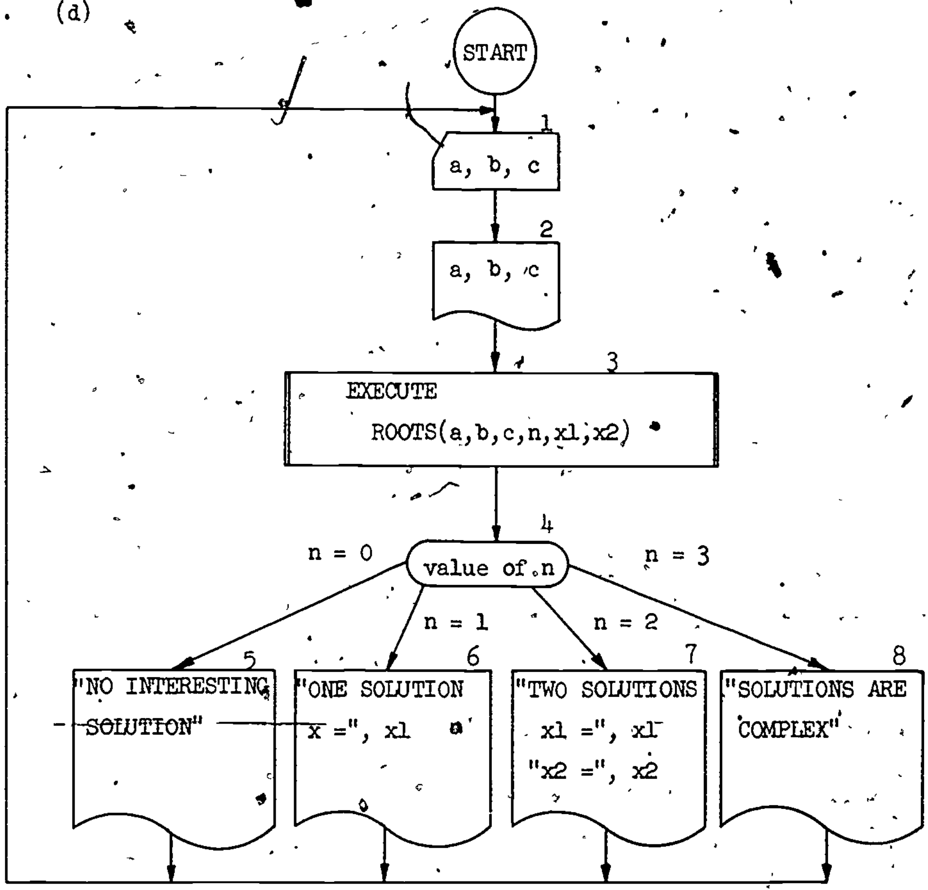
2. (b)


START
1
a, b, c
2
a, b, c
3
EXECUTE
ROOTSA(a,b,c,x1,x2,box5,box6,box8)
7
"TWO SOLUTIONS
x1 =", x1,
"x2 =", x2
5
"NO INTERESTING
SOLUTION"
6
"ONE SOLUTION
x =", x1
8
"SOLUTIONS ARE
COMPLEX"

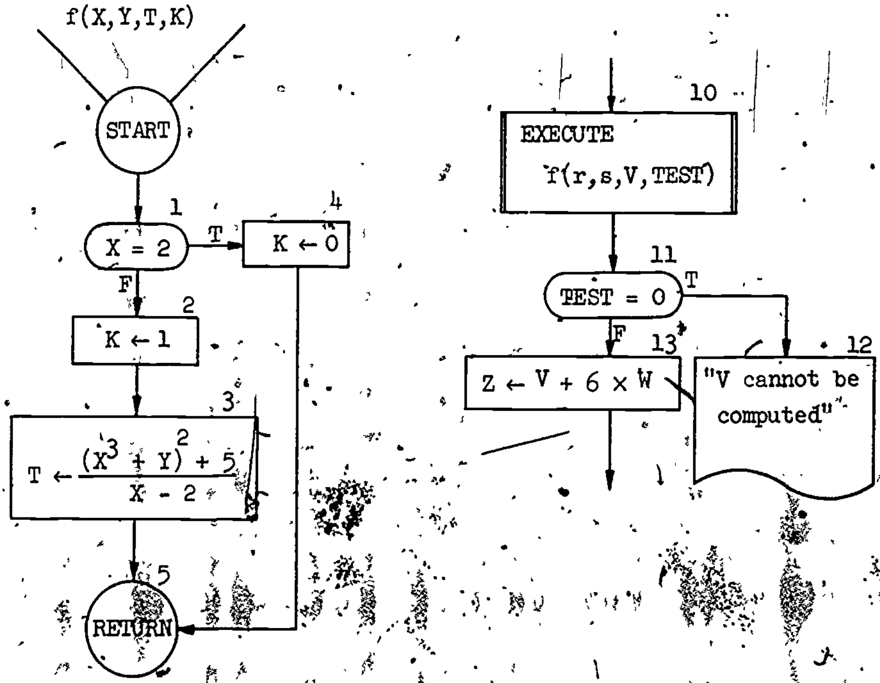
2. (c)


ROOTS(a,b,c,n,x1,x2)
START
1
a ≠ 0
F
T
11
b ≠ 0
F
2
b ≠ 0
T
F
3
DISC ← b² - 4ac
4
DISC > 0
F
T
8
c/a ≥ 0
T
F
6
DISC = 0
F
T
9
n ← 2
x1 ← √-c/a
x2 ← -√-c/a
12
n ← 1
x1 ← - c/b
13
n ← 0
10
n ← 3
7
n ← 1
x1 ← - b/2a
5
n ← 2
x1 ← -b+√DISC / 2a
x2 ← -b-√DISC / 2a
14
RETURN

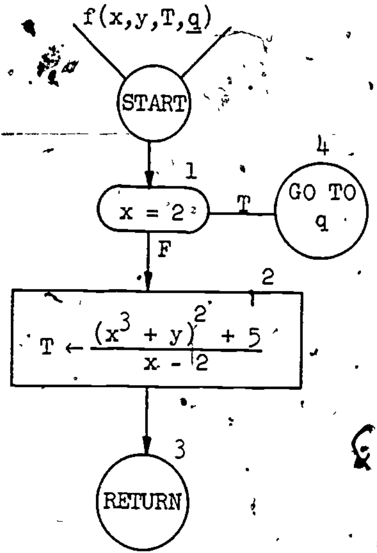
2. (d)

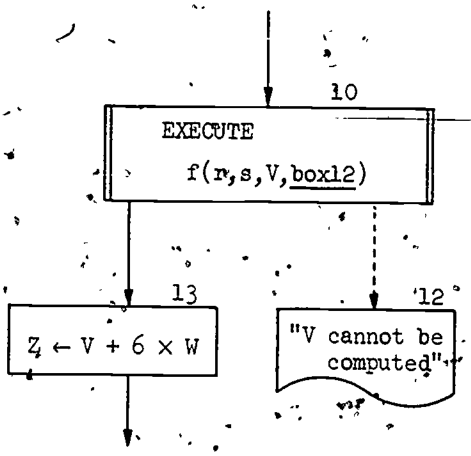

START
1
a, b, c
2
a, b, c
3
EXECUTE
ROOTS(a,b,c,n,x1,x2)
4
value of n
n = 0
n = 1
n = 2
n = 3
5
"NO INTERESTING SOLUTION"
6
"ONE SOLUTION x =", x1
7
"TWO SOLUTIONS x1 =", x1
"x2 =", x2
8
"SOLUTIONS ARE COMPLEX"

3. (a)

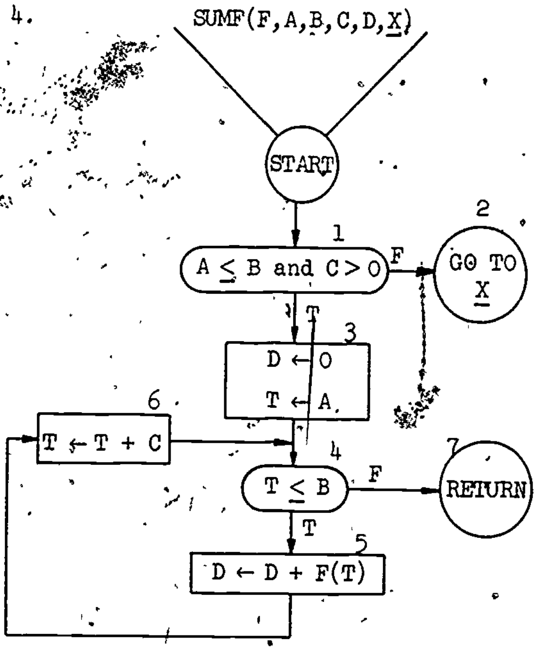

f(X,Y,T,K)
START
1
X = 2
T
4
K ← 0
F
2
K ← 1
3
T ← (X^3 + Y)^2 + 5 / X - 2
5
RETURN
10
EXECUTE
f(r,s,V,TEST)
11
TEST = 0
T
F
13
Z ← V + 6 × W
12
"V cannot be computed"

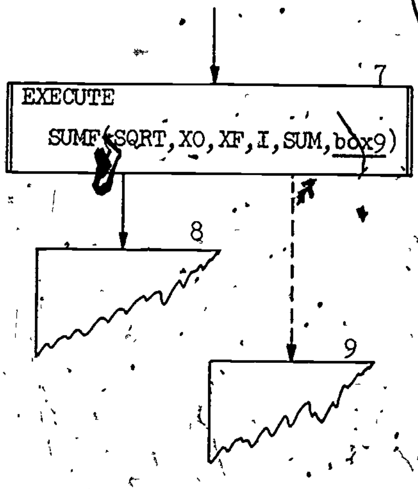
3. (b)


f(x,y,T,q)
START
1
x = 2
T
4
GO TO q
F
2
$T \leftarrow \frac{(x^3 + y)^2 + 5}{x - 2}$
3
RETURN


10
EXECUTE
f(r,s,V,box12)
13
Z ← V + 6 × W
12
"V cannot be computed"

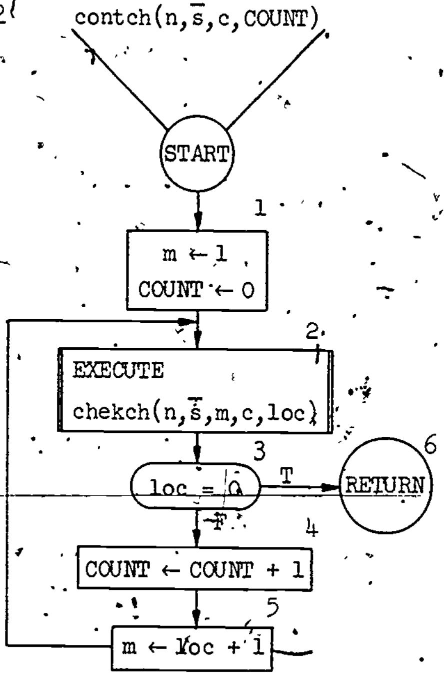
4.


SUMF(F,A,B,C,D,X)
START
1
A ≤ B and C > 0
F
2
GO TO X
T
3
D ← 0
T ← A
6
T ← T + C
4
T ≤ B
F
7
RETURN
T
5
D ← D + F(T)

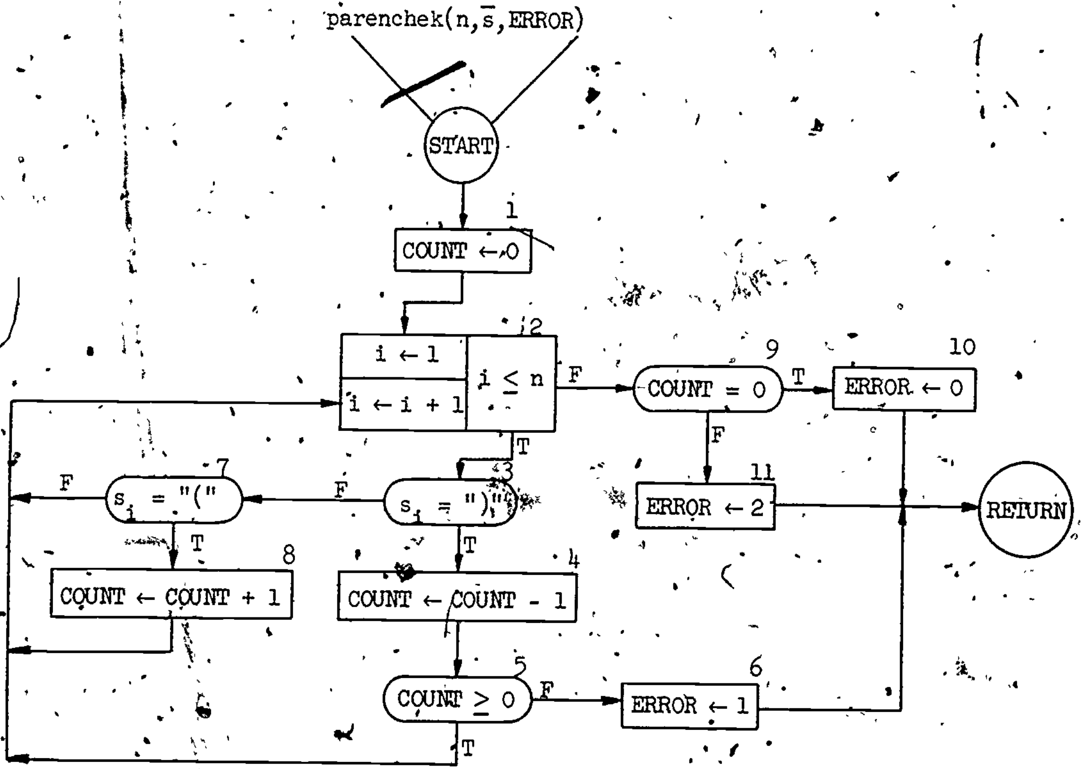

7
EXECUTE
SUMF(SQRT,XO,XF,I,SUM,box9)
8
9

Answers to Exercises 5-6

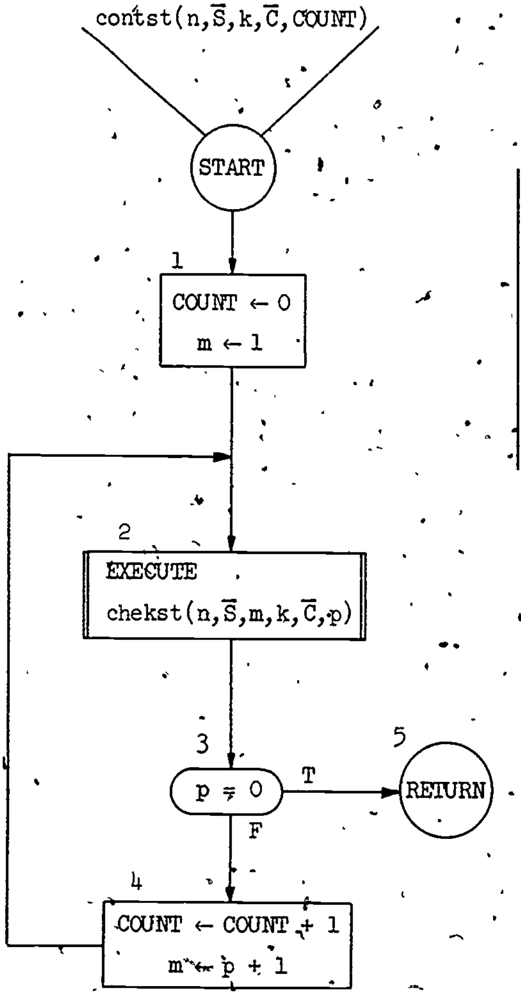
1.



2. Let the value of ERROR equal 0 for a correctly written expression, 1 for a negative counter value and 2 for a nonzero counter value at the end of the scan.



3.



n = length of text shorthand

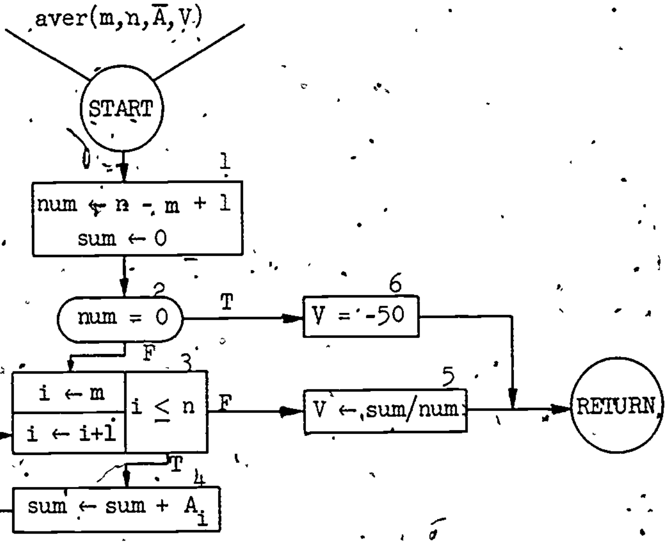
$\overline{S}$ = the string being examined (i.e., $\{s_i, i+1(1)n\}$

k = length of substring

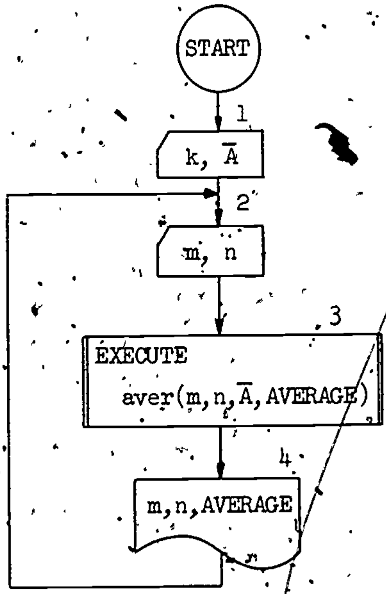
$\overline{C}$ = shorthand for the substring being searched for, i.e., $\{c_i, i+1(1)k\}$

COUNT = the count of occurrences of $\overline{C}$ in $\overline{S}$.

4. (a) For an attempted division by zero, V is set to the unlikely value of -50.



(b)



Chapter T6

APPROXIMATIONS

This chapter is in a way a freak show or chamber of horrors exhibiting samples of many types of difficulties encountered when algorithms are actually carried out in computers. These pathological examples are included not to scare off the reader but rather to make him aware of their existence. Numerical analysts (people who make a specialty of this area) often are able to suggest ways of avoiding the exhibited difficulties in any particular case.

The first section of the chapter concerns itself with representing numbers in a computer while the second briefly reviews what we mean by chopping and rounding to n digits. Section 6-3 illustrates arithmetic on a 3-digit computer. In Section 6-4 we study some of the consequences of the fact that computers have only finite word length. While only a subset of the rational numbers is actually representable in any computer, we often act as though all real numbers were represented. It is interesting and instructive to realize that most of the time what we represent in the computer is only an approximation to the number we have in our mind.

Section 6-5 deals with the consequence that computer arithmetic is not associative; the order of operations may affect the result. Sections 6-6 and 6-7 illustrate difficulties that could arise in solving familiar problems by computers. The last section, 6-8, discusses using a computer to compute the value of a function by methods which themselves are only approximations. The examples are the iterative Newton algorithm for the square root, and computation of sin x by summing terms of a series.

Answers to Exercises 6-1

1. (a) no

(b) yes, 34.2

(c) yes, 3426

(d) no

(e) yes, .078125

(f) yes, 3

(g) yes, 250,827.36

(h) yes, 0

(i) no

(j) no

Answers to Exercises 6-2

1. 49700

2. .00723

3. 42.3

4. 7770

5. 49700

6. .00724

7. 42.4

8. 7780

Answers to Exercises 6-3

1. (a) 467 (b) 46.0

2. (a) 8010 (b) 24.2 (c) .200

3. (a) .185 × 10^{-3} (b) 6.38 (c) .00504

4. (a) 6.87 (b) 1.26 (c) 7.10

5. (a) 9.71 (b) 22.7 (c) 4.76

Answers to Exercises 6-4

1.

```
 .000110011  TWO ①          .0111110
 .000110011      ②          .0001100      ⑥
 ----------                 --------
 .001100110                 .1001010
 .00011001       ③          .000110       ⑦
 ----------                 --------
 .01001100                  .101011
 .0001100        ④          .000110       ⑧
 ----------                 --------
 .0110010                   .110001
 .0001100        ⑤          .000110       ⑨
 ----------                 --------
 .0111110                   .110111
                            .000110       ⑩
                            --------
                            .111101
```

2. $\frac{3}{10}$

```
           .0100101
      ______________
 1010 | 11.0000000
        10 10
        -----
          10000
           1010
           -----
           11000
```

$\frac{7}{10}$

```
            .101100
      ______________
 1010 | 111.000000
        101 0
        -----
         10 000
          1 010
         ------
           1100
           1010
           ----
            1000
```

3. Since multiplying by two in binary just moves the whole string of digits one place to the left, we have

$\frac{2}{10} = .00110011$ $\frac{6}{10} = .100101$

$\frac{4}{10} = .0110011$ obtained by multiplying the binary version of $\frac{2}{10}$ by 2.

$\frac{8}{10} = .110011$

Answers to Exercises 6-8

1. If $a = 0$, the first value of $h = \frac{1}{2}$, and successive values will be $1/2^2$, $1/2^3$, etc. The successive differences $|h-g|$ will be $\left|\frac{1}{2^2} - \frac{1}{2}\right| = \frac{1}{2^2}$, $\left|\frac{1}{2^3} - \frac{1}{2^2}\right| = \frac{1}{2^3}$, etc. The process will terminate when $\left|\frac{1}{2^k} - \frac{1}{2^{k-1}}\right| = \frac{1}{2^k} < \frac{1}{10^4}$. This is equivalent to $10^4 < 2^k$ or $k > \frac{4}{\log 2}$.

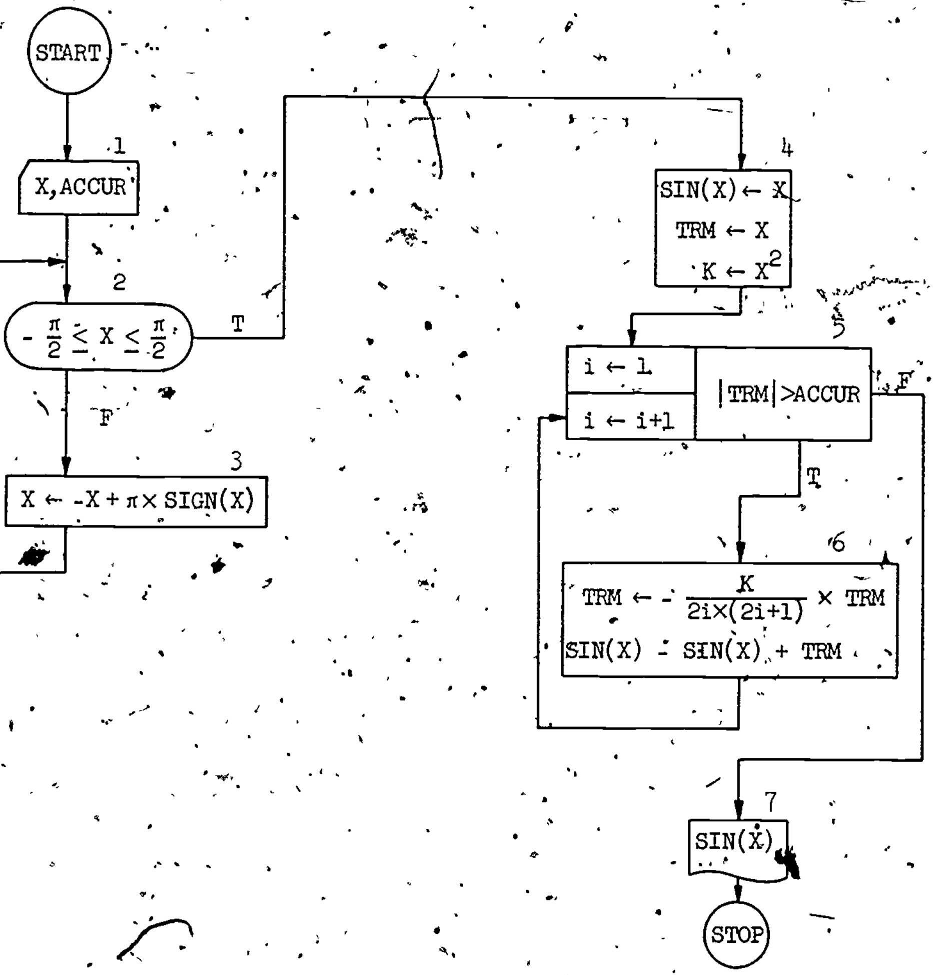
2. If $a < 0$, we know theoretically that there will be trouble. Suppose $a = -2$; we will fill in a little table.

g	$\frac{a}{g}$	$\frac{1}{2}(g + \frac{a}{g})$	$\|h - g\|$
1	-2	$-\frac{1}{2}$	$\frac{1}{2}$
$-\frac{1}{2}$	4	1.75	1.25
1.25	-1.60	-.175	1.42

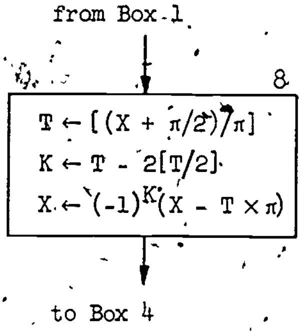
From the last column it is apparent that $|h - g|$ is increasing. The successive approximations are getting worse rather than better.

3. An initial guess related to the size of a would get the algorithm off to a faster start. $\frac{a}{2}$ would be such a guess. However, in this case we must be sure a is not zero since we divide by it in box 2. Since this algorithm is to be used as a reference flow chart, the ability to handle any value of a is more important than saving one iteration of the algorithm. The value of $\frac{a + 1}{2}$ as an initial guess meets both conditions but is more complicated.

4.



In the case that the input value of X is large, many passes through the loop in Boxes 2 and 3 may be required. To avoid this we could replace Boxes 2 and 3 by the mechanism:



Chapter T7

SOME MATHEMATICAL APPLICATIONS

Overview

This chapter is devoted to introducing the use of computer techniques in connection with three problems of the greatest mathematical importance. These problems afford the student his first glimpse of the way computers are really used in applied mathematics.

These problems are: roots of equations seen in Section 7-1; area under a curve in Sections 7-2 and 7-3; and solutions of systems of linear equations in Section 7-4. Thus we meet applications of computers in the areas of theory of equations, integral calculus and linear algebra. The main mathematical areas of computer application which have been omitted are statistics and ordinary and partial differential equations. These latter topics are well beyond the scope of this book.

For further reading we suggest:

(1) Edward Stiefel, An Introduction to Numerical Mathematics, Academic Press.

(2) L. Fox, An Introduction to Numerical Linear Algebra, Oxford Press.

T7-1 Root of an Equation by Bisection

The successive bisection algorithm presented in this section is by no means the only method for finding roots of equations. Newton's method, for example, converges to the root much more rapidly. The bisection algorithm, in essence, picks up one new binary digit of the root on each passage through the loop. Newton's method by contrast will double the number of correct digits (binary or decimal) on each passage through the loop.

We have touched on Newton's method in Section 5-1 as applied to square roots because in this special case we were able to present it without the use of calculus. A general treatment of Newton's method would require the use of differential calculus which we assume the student will not have had. In case your students have had some differential calculus we present later in this section a brief exposition of a Newton's method algorithm which you could present to your students.

The mathematical basis of the bisection algorithm of this section is extremely simple, viz., when a continuous function changes sign in an interval then it has a root in the interval. The number of roots in the interval must either be odd (counting multiplicities) or infinite. In the case that the number of roots is odd and greater than one it is difficult and of little interest to attempt to determine in advance which root the algorithm will converge to. Since the interval [X1,X2] will at all stages contain an odd number of roots, it is relatively easy to see that the method must converge to the first, third, fifth, or seventh, etc., root.

If it is not known in advance that the function is continuous in the given interval, but yet the method converges, then it must converge either to a root or to a point of discontinuity of the function.

In all the preceding discussions of other things which our algorithm will do, one should not lose sight of the fact that the algorithm is primarily intended for the case that there is just one root in the interval. One may even say, for the case that <u>it is known</u> that there is just one root in the interval. One way in which one might know this is by knowing that the derivative of the given function does not change sign in the given interval.

Newton's Method

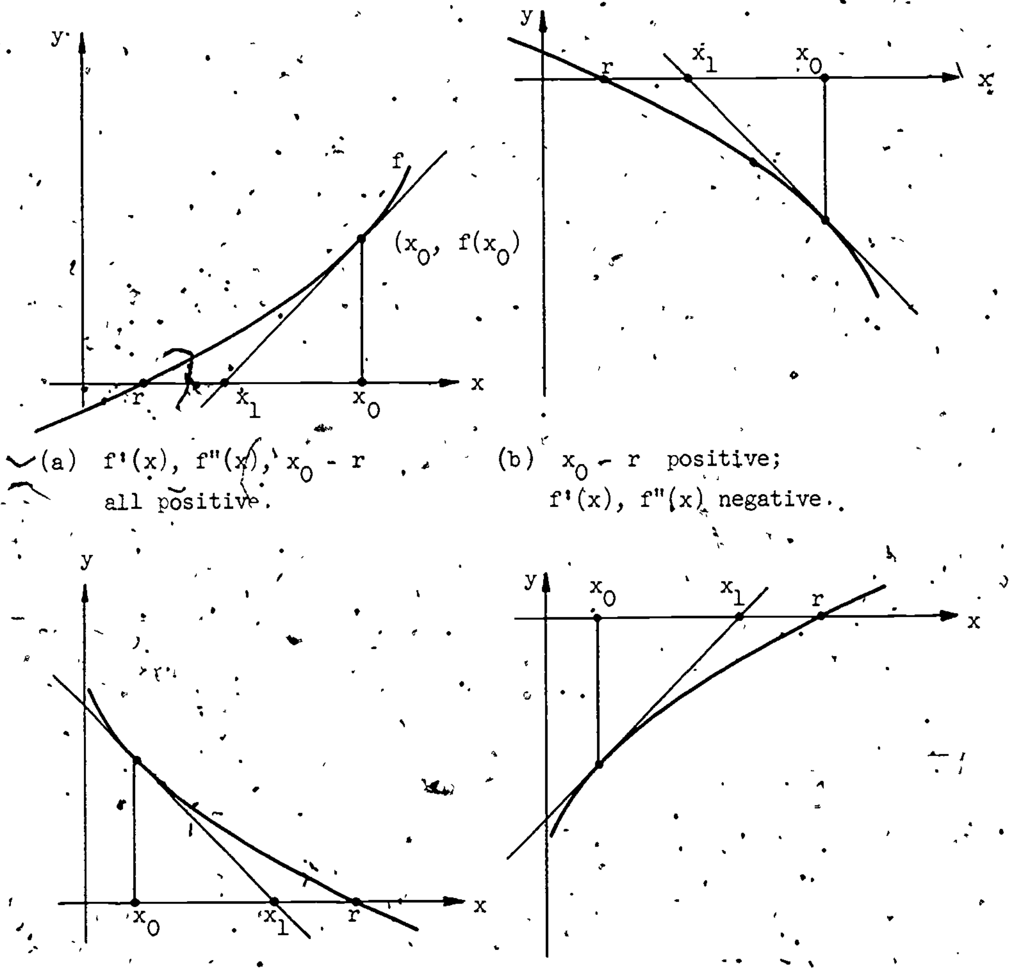
This method was briefly discussed in this Teacher's Commentary in Section 5-1. Here we wish to discuss it in a slightly different way in order to treat the subject of error. This discussion may be found in many calculus books. We will give a condensed description here.

Before embarking on the study of the Newton's method algorithm it would be well to be aware of some of its limitations as compared with the successive bisection algorith. Newton's method converges to the root fantastically faster than the bisection method. If the available binary word length of the machine is ω then the number of iterations of the bisection algorithm loop required for maximum accuracy is $\sim \omega$ while for Newton's method the number of iterations is $\sim \log_2 \omega$. If ω were very large then Newton's method would be enormously superior. Since ω is in practice usually no better than 32, the numbers of iterations are on the order of 32 and 5 respectively, so that the saving is not so very great unless the algorithm is to be used on a large number of problems, or unless a great degree of multiple precision is available, or unless the computation of the functional values is very time consuming.

Furthermore, the use of Newton's method requires guaranteed knowledge of the behavior of the first two derivatives which may not be available in practice. The method is thus only suitable to untabulated functions whose first two derivatives are computable and have certain "nice" properties. The bisection technique imposes no such restrictions.

Still, the Newton's method algorithm has considerable instructive value especially as regards the preliminary use of the bisection technique to obtain reliable bounds on the error. Furthermore, a variant on this method is indispensible in the application of numerical method to differential equations. The analysis which follows should contribute to the understanding of this variant.

Let f be a function having a root at r and let x_0 be a number different from r. Suppose that $f'(x)$ and $f''(x)$ do not change sign or assume the value 0 between r and x_0. Further assume that the constant sign of $f'(x) \cdot f''(x)$ is the same between x_0 and r as the sign of $x_0 - r$. Then the line tangent to the graph of f at the point $(x_0, f(x_0))$ intersects the x-axis at a point x_1 lying between x_0 and r. The four possible cases in which these conditions are met are illustrated below.



(a) $f'(x)$, $f''(x)$, $x_0 - r$ all positive.

(b) $x_0 - r$ positive; $f'(x)$, $f''(x)$ negative.

(c) $x_0 - r$, $f'(x)$ negative; $f''(x)$ positive.

(d) $x_0 - r$, $f''(x)$ negative; $f'(x)$ positive.

Although the method to be derived works in all these cases, we will assume for simplicity in the following discussion that case (a) holds, i.e., $x_0 - r$, $f'(x)$, $f''(x)$ all positive.

The triangle in Figure (a) above yields the ratio

(1) $$\frac{f(x_0)}{x_0 - x_1} = f'(x_0)$$

so that

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

[Since $f''(x) > 0$ for $r \leq x \leq x_0$, then f is convex in this interval so that the tangent line lies below the curve and thus x_1 lies between r and x_0.]

A bound for the error $x_1 - r$ in terms of the original error $x_0 - r$ is next derived.

Using the Law of the Mean we have

$$(2) \qquad \frac{f(x_0) - f(r)}{x_0 - r} = f'(\xi)$$

for some ξ between x_0 and r. Recall that $f(r) = 0$ and rewrite (1) and (2) in the form

$$\frac{1}{x_0 - x_1} = \frac{f'(x_0)}{f(x_0)}, \qquad \frac{1}{x_0 - r} = \frac{f'(\xi)}{f(x_0)}$$

Taking the difference and again applying the law of the mean, this time to $f'(x_0) - f'(\xi)$, we obtain

$$(3) \qquad \frac{1}{x_0 - x_1} - \frac{1}{x_0 - r} = \frac{f'(x_0) - f'(\xi)}{f(x_0)} = \frac{f''(\tau)(x_0 - \xi)}{f(x_0)}$$

where τ lies between x_0 and ξ, hence between x_0 and r.

Getting the left-hand side at (3) over a common denominator and replacing $f(x_0)$ on the right side by $(x_0 - x_1)f'(x_0)$ from (1) we obtain

$$\frac{x_1 - r}{(x_0 - x_1)(x_0 - r)} = \frac{f''(\tau)(x_0 - \xi)}{f'(x_0)(x_0 - x_1)}$$

so that

$$x_1 - r = \frac{f''(\tau)(x_0 - \xi)(x_0 - r)}{f'(x_0)}$$

Hence,

$$|x_1 - r| \leq \frac{M}{N}(x_0 - r)^2.$$

where $M = \max\limits_{r \leq x \leq x_0}|f''(x)|$ and $N = \min\limits_{r \leq x \leq x_0}|f'(x)|$.

Iterating the process by letting x_1 play the role of x_0 we obtain

$$|x_2 - r| \le \frac{M}{N}(x_1 - r)^2 \le \frac{M}{N}\left(\frac{M}{N}(x_0 - r)^2\right)^2$$

$$= \frac{M^3}{N^3}(x_0 - r)^4 .$$

In general,

$$|x_k - r| \le \left(\frac{M}{N}\right)^{2^k - 1}(x_0 - r)^{2^k}$$

$$= \left(\frac{M}{N}(x_0 - r)\right)^{2^k} \Big/ \frac{M}{N} .$$

A disadvantage of this method would seem to be that we need to know the value of r in advance. However, if a is a number known to be on the other side of r from x_0, then

$$|x_k - r| \le \left(\frac{M}{N}|x_0 - a|\right)^{2^k} \Big/ \frac{M}{N} .$$

Now, if $\frac{M}{N}|x_0 - a|$ were known to be fairly small, say less than $\frac{1}{2}$, then the method is seen to coverage at a galloping rate.

A good method of procedure might be to determine successive values of x_0 and a by the bisection method until we have $\frac{M}{N}|x_0 - a| < \frac{1}{2}$ and then switch to the Newton method.

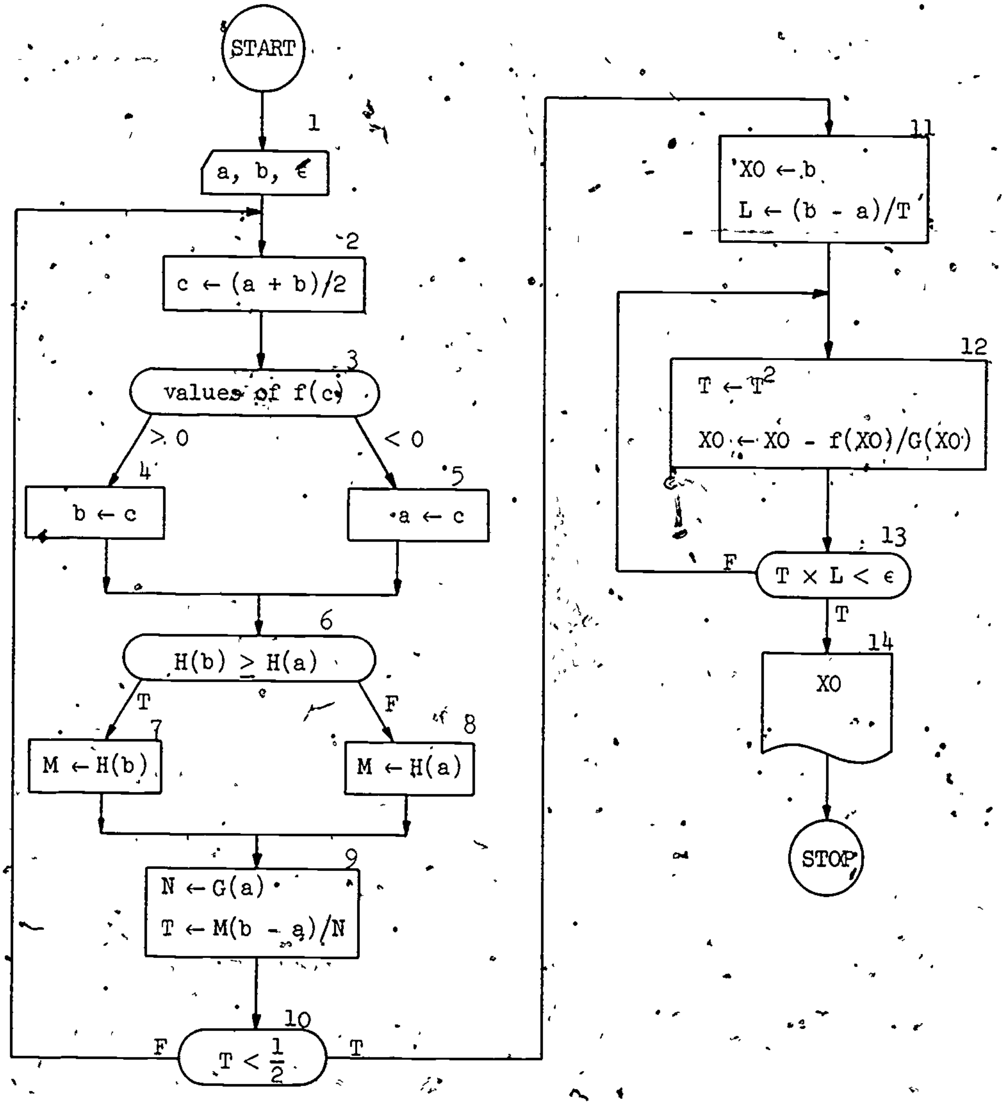
In addition to knowing that the first and second derivatives are positive, we know that the second derivative is monotone, then

$$M \le \max(f''(x_0), f''(a))$$

while

$$N \ge f'(a) .$$

A flow chart for the entire algorithm is given below. We recall all the conditions: $[a,b]$ is an interval in which a root of $f(x) = 0$ is known to lie; $f'(x)$ and $f''(x)$ are known to be positive in $[a,b]$ with $f''(x)$ monotone. [Any of the four cases described earlier can be reduced to the case considered here by suitably replacing $f(x)$ by $-f(x)$ or by $f(-x)$ or by $-f(-x)$.] For convenience we denote $f'(x)$ by $g(x)$ and $f''(x)$ by $h(x)$.



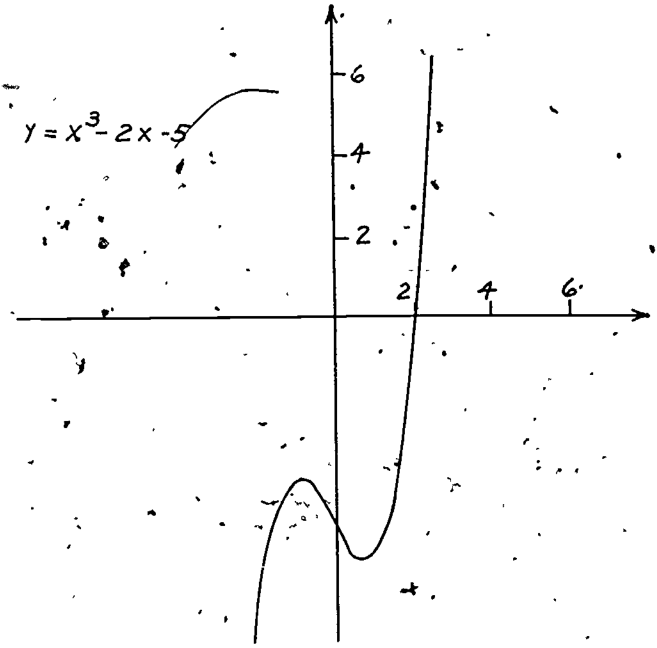
Combined Bisection and Newton's Method Flow Chart

In this flow chart boxes 2 through 10 constitute a modified form of the bisection technique seen in the flow chart of Figure 7-5 of the student text. This variant has as its purpose to beat down the value of T (which is just the $\frac{M}{N}(x_0 - a)$ of the preceding discussion). The test in box 3 of this flow chart is different from the test in box 3 of Figure 7-5 of the student text because we know that f is increasing. Boxes 11 through 13 comprise the Newton's method part of the algorithm. It is striking to see how simple this algorithm is once we know we are in a suitable interval.

Answers to Exercises 7-1 Set A

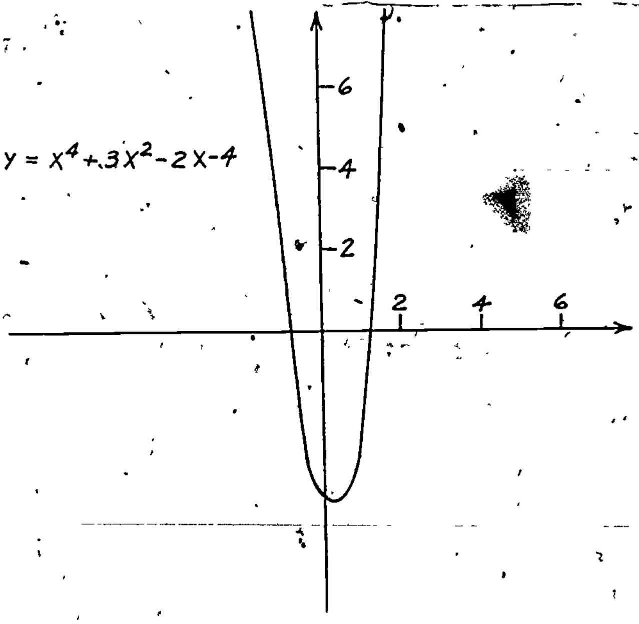
The most difficult of these are 1(e) and 2(c).

1. (a)



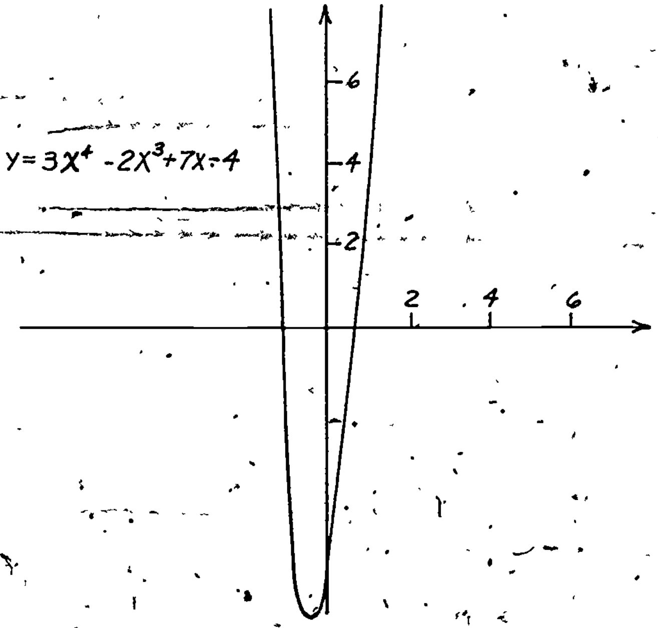
root: near 2.

(b)



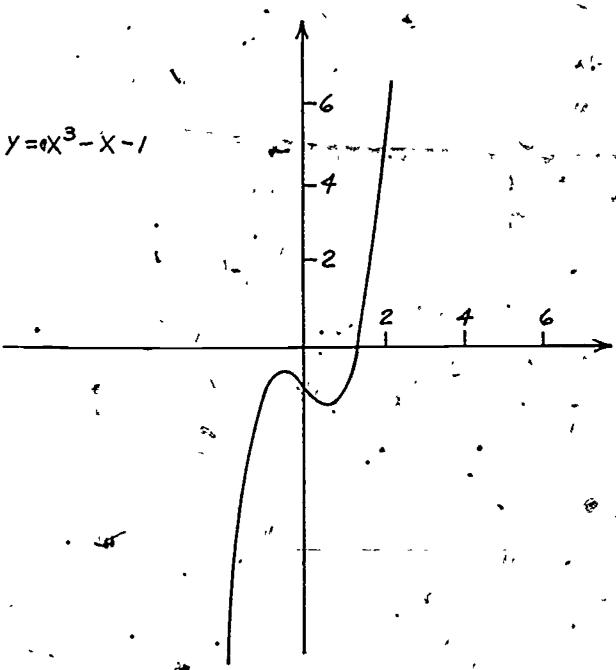
roots: (a) between -1 and 0
(b) between 1 and 2.

(c)



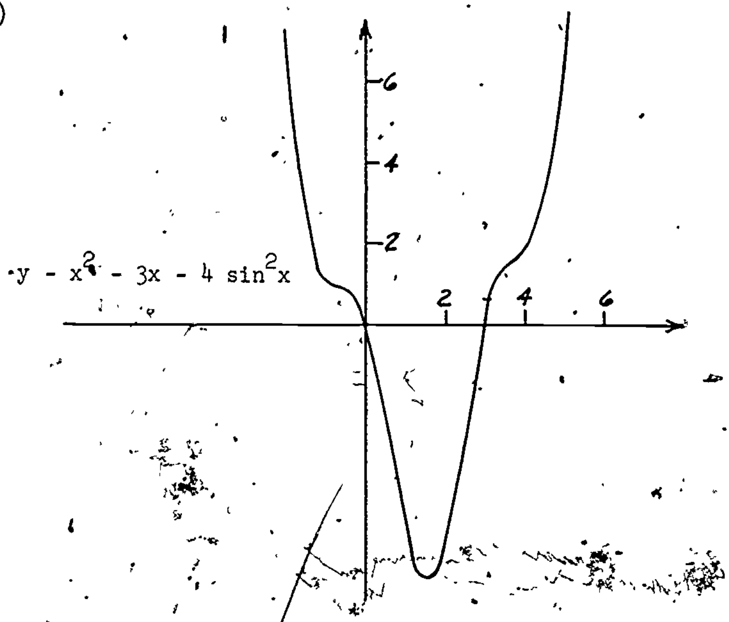
roots: (a) between -2 and -1
(b) between 0 and 1

(d)



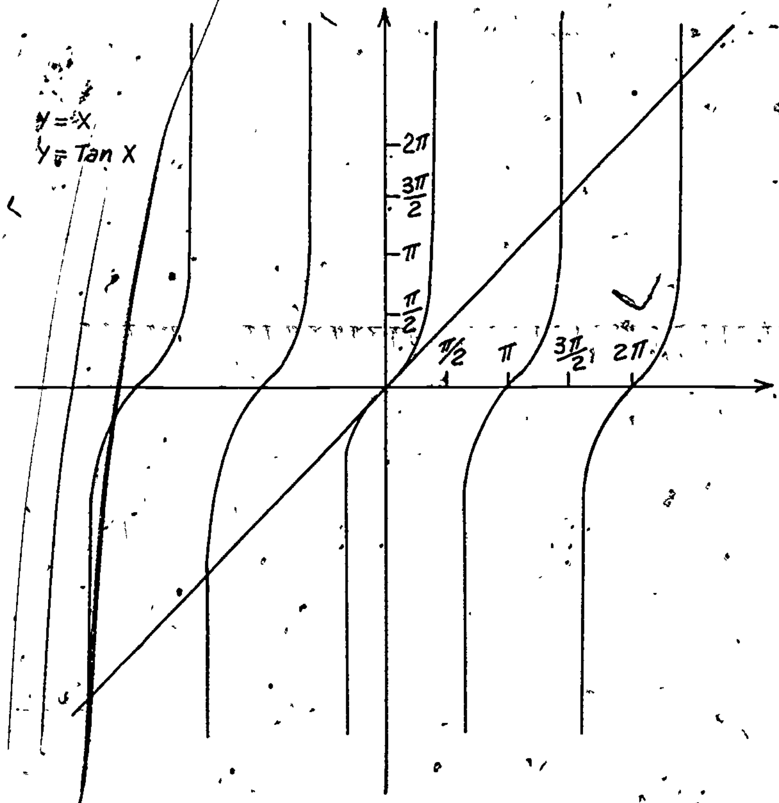
root: between 1 and 2

(e)



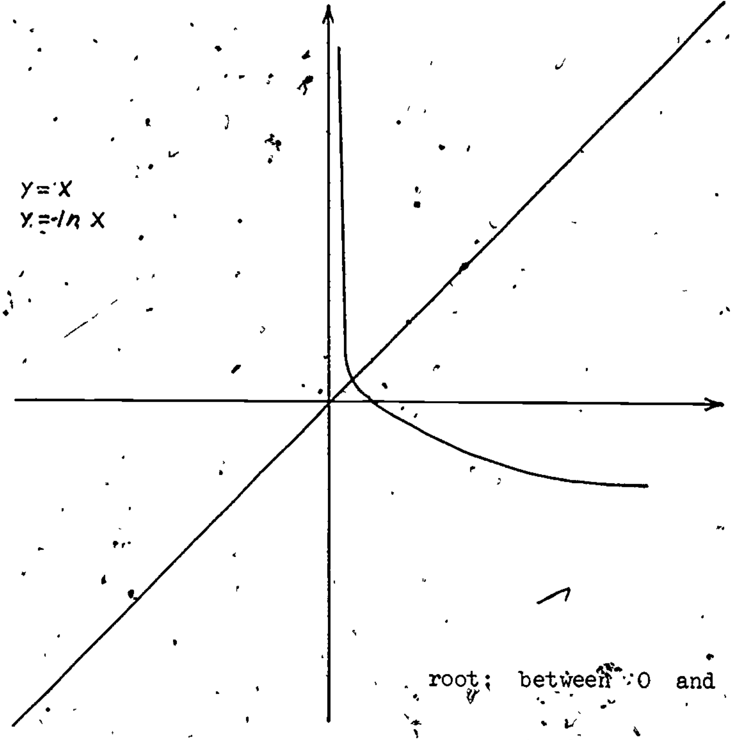
roots: (a) at 0
(b) near 3

2. (a)



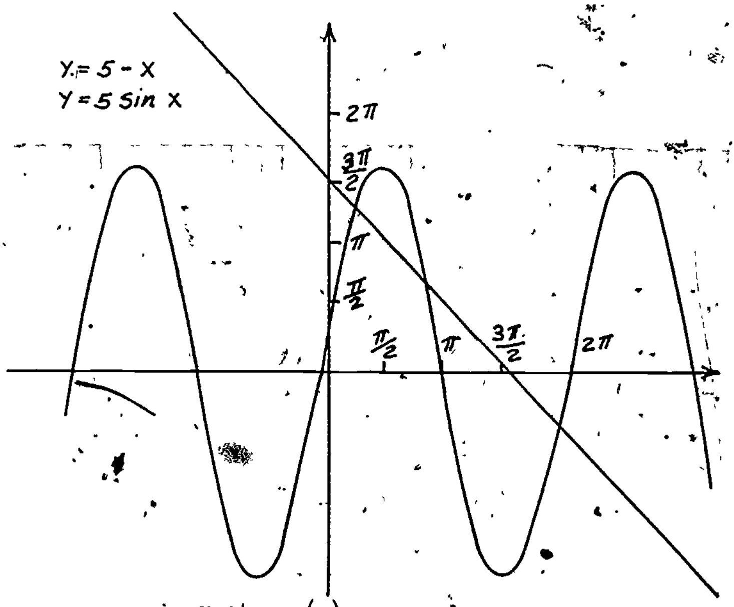
roots: (a) at 0
(b) near $\pm \frac{k}{2}\pi$, k = 3,5,7,...

(b)



root: between 0 and 1

(c)



roots: (a) near 1

(b) between 2 and 3

(c) near 6

Answers to Exercises 7-1 Set B

1. After 4 steps, the root lies in the interval (2.0625, 2.125).

Step	x_1	Sign of $f(x_1)$	x_2	Sign of $f(x_2)$	x_M	Sign of $f(x_M)$
	2	-	3	+	2.5	+
1	2	-	2.5	+	2.25	+
2	2	-	2.25	+	2.125	+
3	2	-	2.125	+	2.0625	-
4	2.0625	-	2.125	+	2.094	

2. After 3 steps, the root is found to lie in the interval (-0.875,-0.75).

Step	x_1	Sign of $f(x_1)$	x_2	Sign of $f(x_2)$	x_M	Sign of $f(x_M)$
	-1	+	0	-	-0.5	-
1	-1	+	-0.5	-	-0.75	-
2	-1	+	-0.75	-	-0.875	+
3	-0.875	+	-0.75	-	-0.8125	

3. After 4 steps, the root lies in the interval (4.375, 4.5).

Step	x_1	Sign of $f(x_1)$	x_2	Sign of $f(x_2)$	x_M	Sign of $f(x_M)$
	3	+	5	-	4	+
1	4	+	5	-	4.5	+
2	4	+	4.5	-	4.25	+
3	4.25	+	4.5	-	4.375	+
4	4.375	+	4.5	-	4.4375	

Answers to Exercises 7-1 Set C

1. For $\epsilon = 0.1$ the root is 1.34.

$f(x) = x^3 - x - 1 = 0$

Step	x_1	Sign of $f(x_1)$	x_2	Sign of $f(x_2)$	x_m	Sign of $f(x_m)$	$\lvert x_1-x_2\rvert$
	0	-	2	+	1	-	2
1	1	-	2	+	1.5	+	1
2	1	-	1.5	+	1.25	-	0.5
3	1.25	-	1.5	+	1.375	+	0.25
4	1.25	-	1.375	+	1.3125	-	0.125
5	1.3125	-	1.375	+	1.34375	+	0.0625

2. For $\epsilon = 0.15$, the root is 0.606.

$f(x) = x + \ell n x = 0$

Step	x_1	Sign of $f(x_1)$	x_2	Sign of $f(x_2)$	x_m	Sign of $f(x_m)$	$\lvert x_1-x_2\rvert$
	0.1	-	1	+	0.55	-	0.9
1	0.55	-	1	+	0.775	+	0.45
2	0.55	-	0.775	+	0.6625	+	0.225
3	0.55	-	0.6625	+	0.60625	+	0.1125

3. For $\epsilon = 0.4$, the root is 0.875.

$f(x) = 5 - x - 5 \sin x = 0$

Step	x_1	Sign of $f(x_1)$	x_2	Sign of $f(x_2)$	x_m	Sign of $f(x_m)$	$\lvert x_1-x_2\rvert$
	0	+	2	-	1	-	2
1	0	+	1	-	0.5	+	1
2	0.5	+	1	-	0.75	+	0.5
3	0.75	+	1	-	0.875	+	0.25

4. The root is 2. $f(x) = x^3 - 3x - 2 = 0$

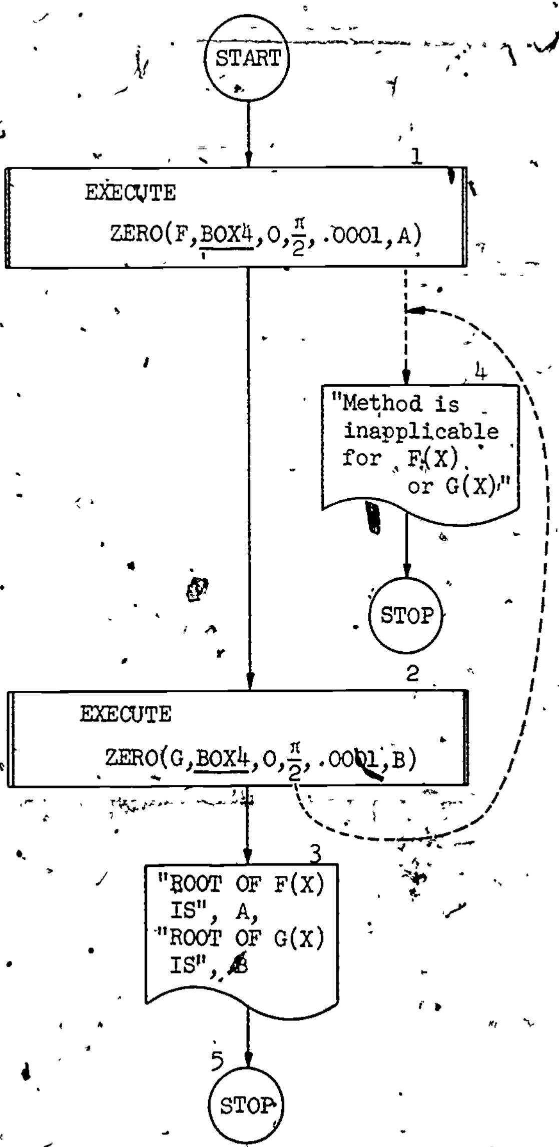
5. $f(x) = x^3 - 2x^2 - 13x - 10 = 0$

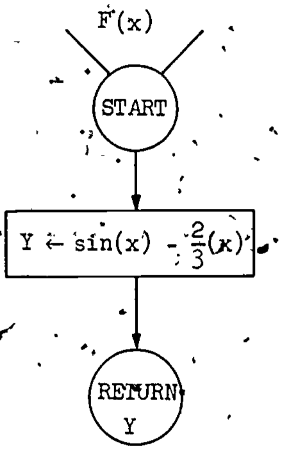
"Method inapplicable" would be printed.

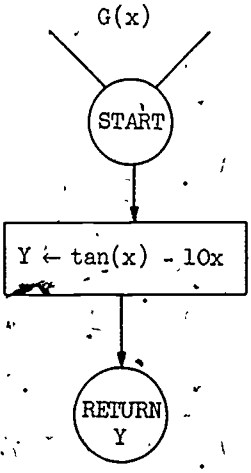
Answers to Exercises 7-1 Set D

1. Let $F(x) = \sin x - \frac{2}{3}x$

$G(x) = \tan x - 10x$

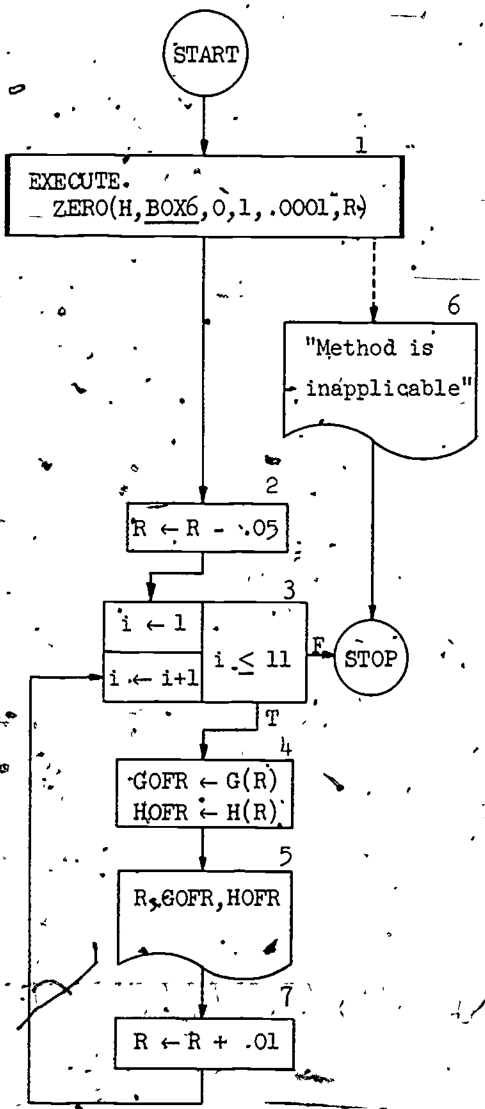

START
1
EXECUTE
ZERO(F,BOX4,0,$\frac{\pi}{2}$,.0001,A)
4
"Method is inapplicable for F(X) or G(X)"
STOP
2
EXECUTE
ZERO(G,BOX4,0,$\frac{\pi}{2}$,.0001,B)
3
"ROOT OF F(X) IS", A, "ROOT OF G(X) IS", B
5
STOP

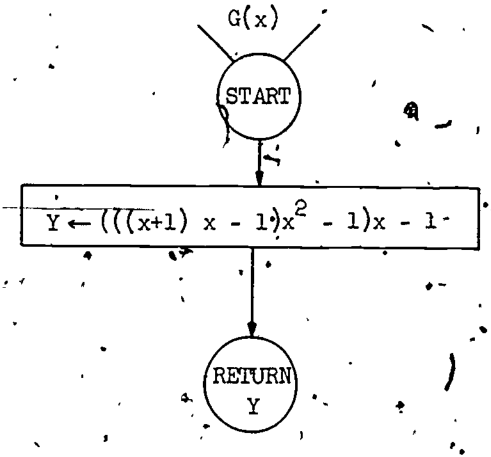


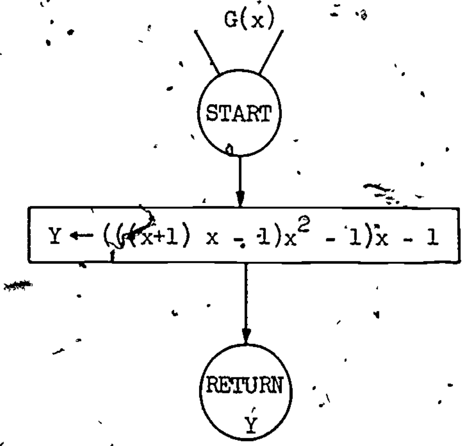


Comment: Box 4 should never be executed--but "just in case", we make provision for printing a message. In computer work clerical errors like key punch errors, or a "bug" in the zero procedure, could cause the computer to take the alternate exit. If it happens, we want to print a message so we know it's happened. We take the same approach in giving similar print boxes in the solutions to the other exercises in this set.

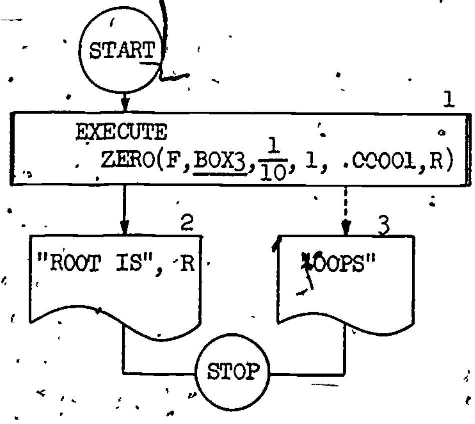
2.

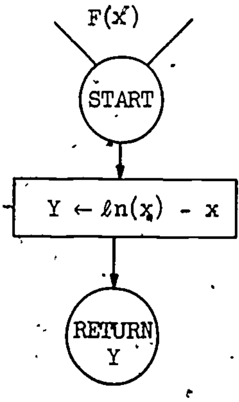






3. The student should be able to see that the root lies in the interval $(\frac{1}{10},1)$.



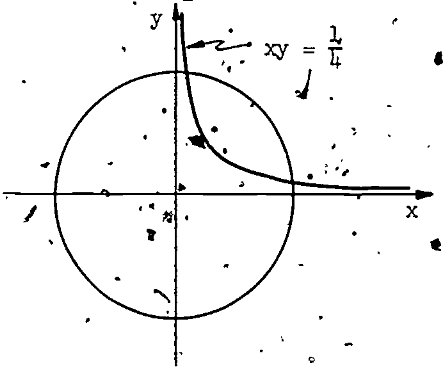


4. Equation for the orbit is $x^2 + y^2 = 1$.

Note that in the first quadrant, $y = \sqrt{1 - x^2}$.

(a) Intersections with the hyperbola $xy = \frac{1}{4}$ $0 \leq x \leq \frac{1}{2}$ are obtained by solving:

$$\sqrt{1 - x^2} = \frac{1}{4x} .$$



Let $f(x) = x \cdot \sqrt{1 - x^2} - \frac{1}{4} = 0$

a) interval $0 < x \leq \frac{1}{2}$

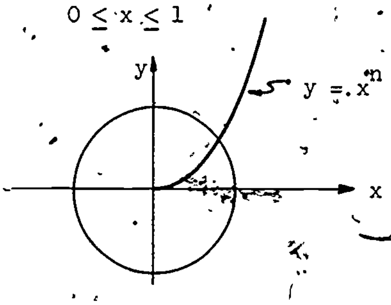
b) interval $\frac{1}{2} < x \leq 1$

(b) Intersection with the power curves

$$y = x^n , \qquad n = 1,2,3,4,5 \qquad 0 \leq x \leq 1$$

in the first quadrant is expressed as

$$\sqrt{1 - x^2} = x^n .$$



Let $G1(x) = \sqrt{1 - x^2} - x$ (solution in this case is $x = \frac{\sqrt{2}}{2}$.)

$G2(x) = \sqrt{1 - x^2} - x^2$

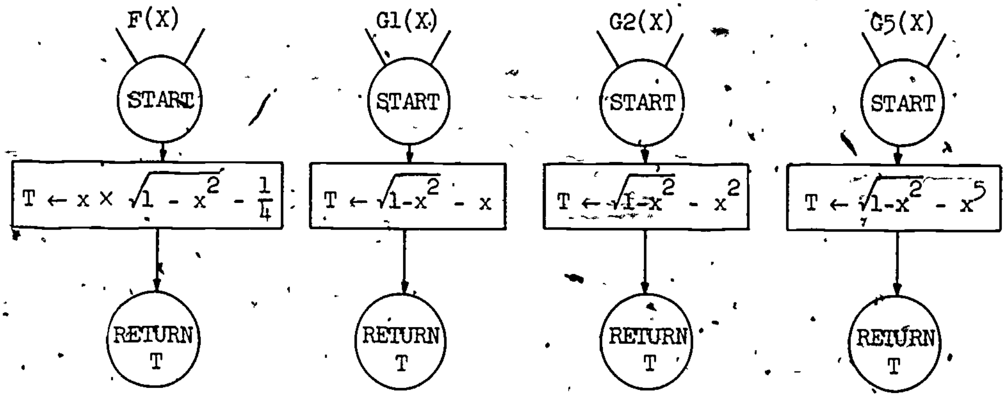
.

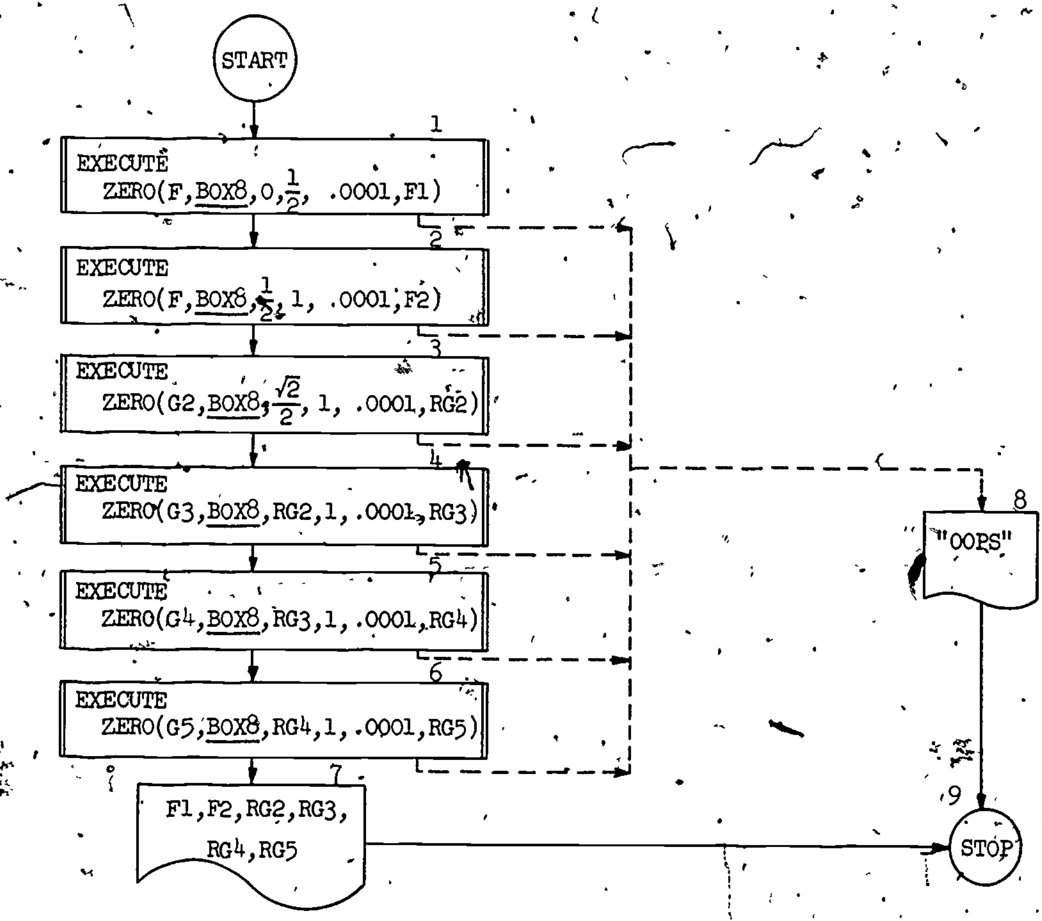
.

.

$G5(x) = \sqrt{1 - x^2} - x^5$

The required flow charts:

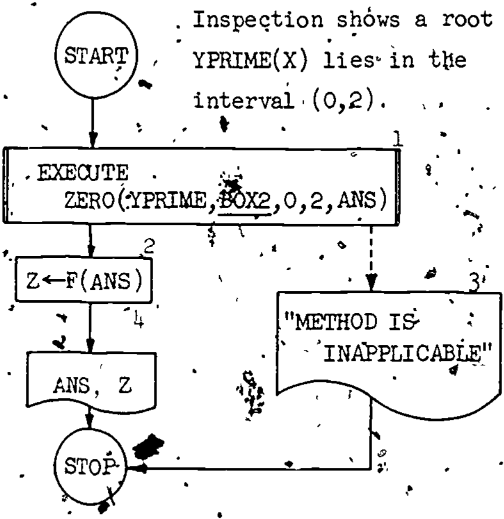


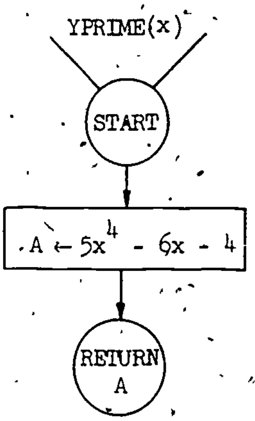

START
1
EXECUTE ZERO(F,BOX8,0,$\frac{1}{2}$, .0001,F1)
2
EXECUTE ZERO(F,BOX8,$\frac{1}{2}$,1, .0001,F2)
3
EXECUTE ZERO(G2,BOX8,$\frac{\sqrt{2}}{2}$, 1, .0001,RG2)
4
EXECUTE ZERO(G3,BOX8,RG2,1,.0001,RG3)
5
EXECUTE ZERO(G4,BOX8,RG3,1,.0001,RG4)
6
EXECUTE ZERO(G5,BOX8,RG4,1,.0001,RG5)
7
F1,F2,RG2,RG3, RG4,RG5
8
"OOPS"
9
STOP

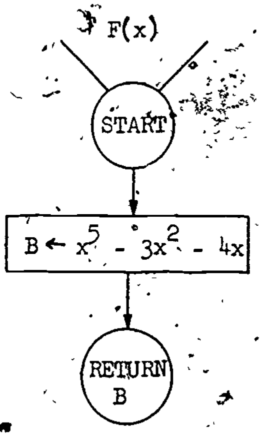
Comment: We do not bother calling on the zero procedure for computing the root of G1(x) since the solution is obviously $\sqrt{2}/2$. Notice that we can use the root of G1(x) as the lower limit for the beginning bisect interval in computing the root of G2(x). In general, the root of Gi(x) becomes the lower limit of the interval search for the root of G(i+1)(x).

5. $\frac{dy}{dx} = 5x^4 - 6x - 4 = 0 = \text{YPRIME(X)}$

Inspection shows a root of YPRIME(X) lies in the interval (0,2).







6. w = 15 feet

T7-2 Area Under a Curve: An Example: $y = 1/x$ between $x = 1$ and $x = 2$

This section constitutes the student's introduction to the subject of integration, although we have deliberately avoided the use of that word. Such a projection into the future would best (we feel) be casually passed off by the teacher.

There are compelling reasons for believing that integration is best introduced via computation. In most textbook approaches to integration very little time is devoted to computing integrals from the definition since most examples are too difficult for hand computation. Instead, the texts head with all possible dispatch for the Fundamental Theorem of Calculus. The effect on the student is to leave him thinking of the integral only as the anti-derivative--an unfortunate viewpoint for most applications. With computational techniques available, the student can program, or at least flow chart, a wide variety of integrals before meeting with the Fundamental Theorem of Calculus.

Furthermore, the computational method is more closely related to real life where anti-derivatives can hardly ever be found. It is a sort of a miracle that for certain elementary functions the integral can be calculated explicitly. This miracle is of fundamental importance in mathematics. But, it should not bar us from approximating integrals when no miracle occurs.

The numerical integration technique developed in this section is the trapezoidal rule. It produces an approximation of the integral regardless of whether the function is everywhere positive in the given interval. Of course, the interpretation of the integral in cases in which the function is not everywhere positive, would be very difficult and distracting for students at this stage.

To be sure, there are methods of approximating the integral which converge much more rapidly to the integral than the trapezoid rule. The most famous of these is Simpson's Rule, obtained by approximating the function by parabolas rather than line segments. (See almost any Calculus text.) The approximation is given by

$$\int_a^b f(x)dx \cong \frac{1}{3}(f(a) + 4f(a + h) + 2f(a + 2h) + 4f(a + 3h) + 2f(a + 4h) + 4f(a + 5h) + \dots + 2f(a + (2n-2)h) + 4f(a + (2n-1)h) + f(b))$$

where $h = \frac{b-a}{2n}$. The algorithm is simply flowcharted, but its justification is beyond the scope of this text.

In the text there has not been included a discussion of a bound on the error in using the trapezoid rule. We will next discuss this subject here.

Letting $h = \frac{b-a}{2n}$ we see that the bisection method gives the approximate value, A_k, of the area over the k^{th} subinterval as

$$A_k = \frac{f(a + (k-1)h) + f(a + kh)}{2} h$$

while the actual area, I_k, is

$$I_k = \int_{a + (k-1)h}^{a + kh} f(x)dx .$$

For some ξ and some η in $[a + (k-1)h, a + kh]$

$$\frac{f(a + (k-1)h) + f(a + kh)}{2} = f(\xi)$$

and

$$\int_{a + (k-1)h}^{a + kh} f(x)dx = hf(\eta)$$

by the mean value theorem for integrals. Thus,

$$\begin{aligned} |I_k - A_k| &= |f(\xi) - f(\eta)|h \\ &= \left|\frac{f(\xi) - f(\eta)}{\xi - \eta}\right| h|\xi - \eta| . \end{aligned}$$

By the law of the mean

$$\frac{f(\xi) - f(\eta)}{\xi - \eta} = f'(\tau)$$

for some τ between ξ and η. Using the fact tnat $\tau\xi - \eta\tau \le h$ we have

$$|I_k - A_k| \le M_1 h^2$$

where $M_1 = \max_{a \le x \le b} f'(x)$. Multiplying by the number of subintervals, $\frac{b-a}{h}$, we get an upper bound for the difference in the total areas

$$|I - A| \le M_1(b - a)h.$$

In the n^{th} iteration of the bisection process, the length, h, of the subintervals is $\frac{1}{2^n}$ so that the error will be less than

$$\frac{M_1(b - a)}{2^n} .$$

If we know a bound, M_2, for the second derivative of f in the interval $[a,b]$, we can get the much better bound for the error

$$\frac{M_2(b - a)h^2}{12} = \frac{M_2(b - a)}{12 \cdot 4^n} .$$

For Simpson's rule the bound on the error is

$$\frac{M_2(b - a)h^4}{180} = \frac{M_2(b - a)}{180 \cdot 16^n} .$$

For a calculation of these bounds for the error see, for example, R. Courant, Calculus I, Interscience.

Answers to Exercises 7-2 Set A

1. T_1 = area of trapezoid ACQP plus area of trapezoid CBRQ

$$= \frac{1}{2} \times \frac{1}{2}[f(1) + f(\tfrac{3}{2})] + \frac{1}{2} \times \frac{1}{2}[f(\tfrac{3}{2}) + f(2)]$$

$$= \frac{1}{4}(1 + \frac{2}{3}) + \frac{1}{4} \times (\frac{2}{3} + \frac{1}{2})$$

$$= \frac{1}{4}(1 + \frac{4}{3} + \frac{1}{2}) = \frac{1}{4} \times \frac{6 + 8 + 3}{6} = \frac{17}{24}$$

2. T_0 = area of trapezoid ABRP = $\frac{3}{4}$

T_1 = area of trapezoid ACQP plus area of trapezoid CBRQ = $\frac{17}{24}$.

$T_0 - T_1$ = area of triangle PQR

$$= \frac{3}{4} - \frac{17}{24} = \frac{1}{24} .$$

Answers to Exercises 7-2 Set B

1. T_2 = area of trapezoid ADSP plus area of trapezoid DCQS plus area of trapezoid CEVQ plus area of trapezoid EBRV

$$= \frac{1}{4}\times\frac{1}{2}[f(1) + f(\tfrac{5}{4})] + \frac{1}{4}\times\frac{1}{2}[f(\tfrac{5}{4}) + f(\tfrac{3}{2})] + \frac{1}{4}\times\frac{1}{2}[f(\tfrac{3}{2}) + f(\tfrac{7}{4})] + \frac{1}{4}\times\frac{1}{2}[f(\tfrac{7}{4}) + f(2)]$$

$$= \frac{1}{8}[1 + \frac{4}{5}] + \frac{1}{8}[\frac{4}{5} + \frac{2}{3}] + \frac{1}{8}[\frac{2}{3} + \frac{4}{7}] + \frac{1}{8}[\frac{4}{7} + \frac{1}{2}]$$

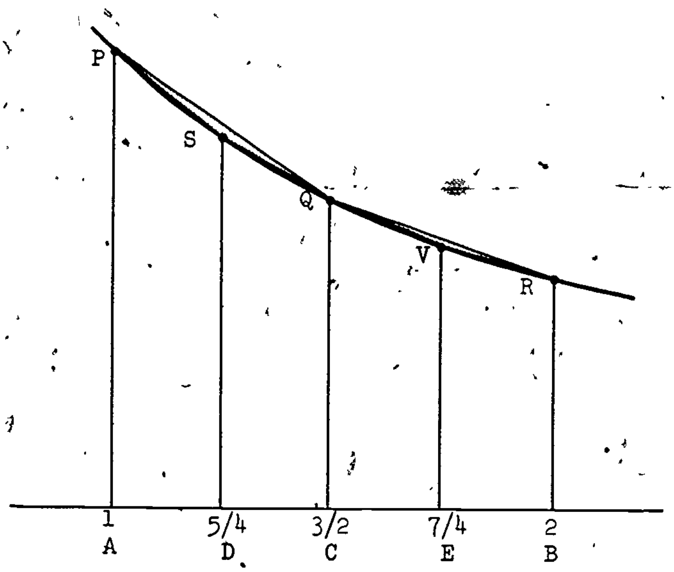
$$= \frac{1}{8}[1 + \frac{8}{5} + \frac{4}{3} + \frac{8}{7} + \frac{1}{2}] = \frac{1}{8} \times \frac{210 + 336 + 280 + 240 + 105}{210}$$

$$= \frac{1171}{1680}$$

2. $$T_1 - T_2 = \frac{17}{24} - \frac{1171}{1680} = \frac{1190}{1680} - \frac{1171}{1680}$$

$$= \frac{19}{1680} \cong .01131$$

3.



T_1 = area of trapezoid ACQP plus area of trapezoid CBRQ.

T_2 = area of trapezoid ADSP plus area of trapezoid DCQS plus area of trapezoid CEVQ plus area of trapezoid EBRV

$T_1 - T_2$ = (area ACQP minus area ADSP minus area DCQS) plus (area CBRQ minus area CEVQ minus area EBRV)

= area triangle PSQ plus area triangle QVR

Answers to Exercises 7-2 Set C

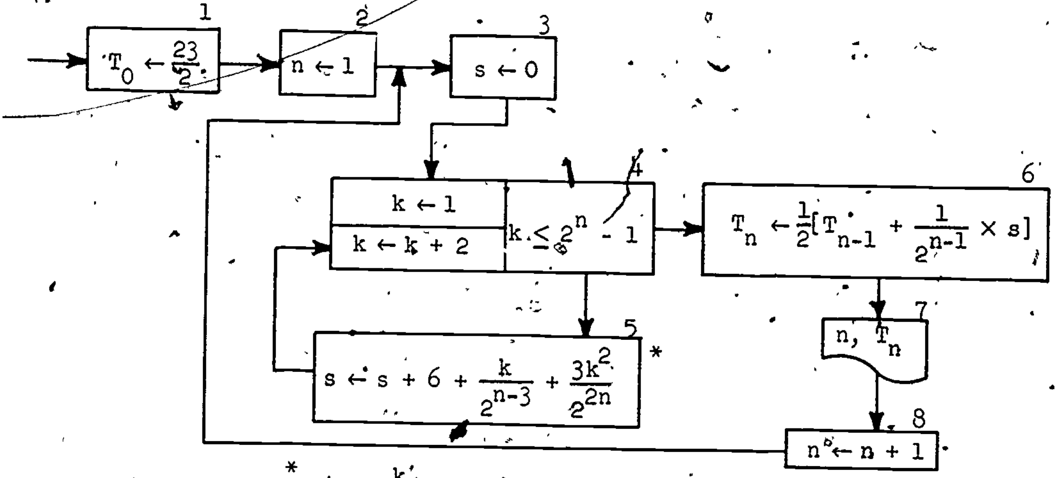
Flow chart to 7(b) should be saved; it will be referred to in the language text.

1. Abscissa values are 1, 17/16, 9/8, 19/16, 5/4, 21/16, 11/8, 23/16, 3/2, 25/16, 13/8, 27/16, 7/4, 29/16, 15/8, 31/16, 2.

2. The number doubles.

3. The abscissa values are $1, \frac{3}{2}, 2, \frac{5}{2}, 3, \frac{7}{2}, 4, \frac{9}{2}, 5$.

4.


1
$T_0 \leftarrow \frac{23}{2}$
2
$n \leftarrow 1$
3
$s \leftarrow 0$
$k \leftarrow 1$
$k \leftarrow k + 2$
4
$k \leq 2^n - 1$
5
$s \leftarrow s + 6 + \frac{k}{2^{n-3}} + \frac{3k^2}{2^{2n}}$
*
6
$T_n \leftarrow \frac{1}{2}[T_{n-1} + \frac{1}{2^{n-1}} \times s]$
7
n, T_n
8
$n \leftarrow n + 1$

$^* f(1 + \frac{k}{2^n})$

$$f(x) = 3x^2 + 2x + 1$$

$$= 3(1 + \frac{k}{2^n})^2 + 2(1 + \frac{k}{2^n}) + 1$$

$$= 3(1 + \frac{2k^2}{2^n} + \frac{k^2}{2^{2n}}) + 2 + \frac{2k^2}{2^n} + 1$$

$$= 6 + \frac{8k}{2^n} + \frac{3k^2}{2^{2n}}$$

$$= 6 + \frac{k}{2^{n-3}} + \frac{3k^2}{2^{2n}}$$

5. The purpose of this problem is to help the student to understand the flow chart of Figure 7-16 by asking leading questions.

Solution:

(a) $T_0 = \frac{1 + 4}{2} = \frac{5}{2}$

(b) Enter twice. Exit once to box 5, once to box 6.

(c) $T_1 = \frac{19}{8}$

(d) $2^n - 1 = 7$. You enter for $k = 1, 3, 5, 7,$ and 9.

(e) $k = 9$ when the test fails. At that time you just calculated $f(1 + \frac{7}{8})$ and added it to S.

$$S = f(1 + \tfrac{1}{8}) + f(1 + \tfrac{3}{8}) + f(1 + \tfrac{5}{8}) + f(1 + \tfrac{7}{8})$$
$$= (\tfrac{9}{8})^2 + (\tfrac{11}{8})^2 + (\tfrac{13}{8})^2 + (\tfrac{15}{8})^2$$
$$= \tfrac{1}{64}(81 + 121 + 169 + 225)$$
$$= \tfrac{596}{64} = \tfrac{149}{16}$$

(f) $k \leq 2^{10} - 1 = 1023$. You enter $\frac{1024}{2} = 512$ times and go on to box 5. The 513th time you go on to box 6. So you enter 513 times in all.

(g) $T_0 = 2.50$
$T_1 = 2.36$
$T_2 = 2.34$
$T_2 - T_1 = .02$

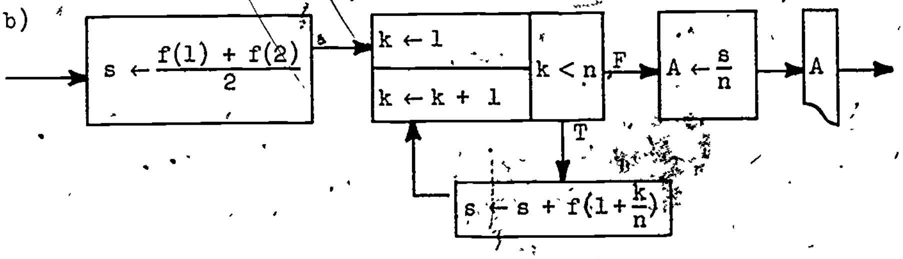
$n = 2$ when box 9 is entered.

6. (a) $A = \frac{1}{2n}[f(1) + 2f(1 + \frac{1}{n}) + 2f(1 + \frac{2}{n}) + \ldots + 2f(1 + \frac{n-1}{n}) + f(2)]$

or

$A = \frac{1}{n}[\frac{f(1) + f(2)}{2} + \sum_{i=1}^{n-1} f(1 + \frac{i}{n})]$

(b)


$s \leftarrow \frac{f(1) + f(2)}{2}$
$k \leftarrow 1$
$k \leftarrow k + 1$
$k < n$
F
T
$A \leftarrow \frac{s}{n}$
A
$s \leftarrow s + f(1+\frac{k}{n})$

7. (b) $T_0 = \frac{1}{2}(8)[f(1) + f(9)] = 4[f(1) + f(9)] = 4(1 + \frac{1}{9}) = \frac{40}{9} = 4.444$

(c) $T_1 = \frac{1}{2}(4)[f(1) + 2f(5) + f(9)] = 2[f(1) + 2f(5) + f(9)]$

$= 2(1 + \frac{2}{5} + \frac{1}{9}) = \frac{136}{45} = 3.022$

(d) $T_2 = \frac{1}{2}(2)[f(1) + 2f(3) + 2f(5) + 2f(7) + f(9)]$

$= 1 + \frac{2}{3} + \frac{2}{5} + \frac{2}{7} + \frac{1}{9} = \frac{776}{315} = 2.463$

(e) $T_3 = \frac{1}{2}(1)[f(1) + 2f(2) + 2f(3) + 2f(4) + 2f(5) + 2f(6) + 2f(7) + + 2f(8) + f(9)]$

$= \frac{1}{2}(1 + 1 + \frac{2}{3} + \frac{1}{2} + \frac{2}{5} + \frac{1}{3} + \frac{2}{7} + \frac{1}{4} + \frac{1}{9})$

$= \frac{5729}{2520} = 2.273$

(f) $T_1 = \frac{1}{2}[T_0 + 8f(5)]$

$T_2 = \frac{1}{2}[T_1 + 4f(3) + 4f(7)]$

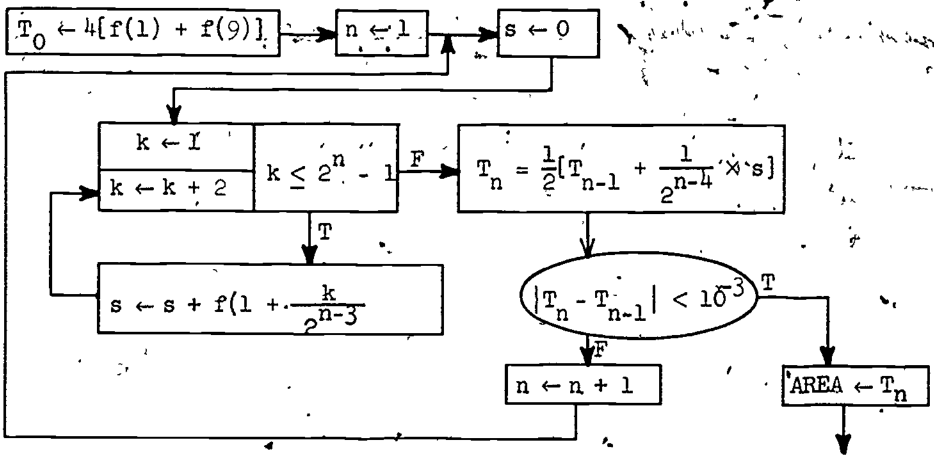
$T_3 = \frac{1}{2}[T_2 + 2f(2) + 2f(4) + 2f(6) + 2f(8)]$

(g) $T_4 = \frac{1}{2}[T_3 + f(\frac{3}{2}) + f(\frac{5}{2}) + f(\frac{7}{2}) + f(\frac{9}{2}) + f(\frac{11}{2}) + f(\frac{13}{2}) + f(\frac{15}{2}) + f(\frac{17}{2})]$

$$T_n = \frac{1}{2}T_{n-1} + \frac{8}{2^n} \sum_{\substack{k=1 \\ k \text{ odd}}}^{2^n} f(1 + \frac{8k}{2^n})$$

$$= \frac{1}{2}T_{n-1} + \frac{1}{2^{n-3}} \sum_{\substack{k=2 \\ k \text{ odd}}}^{2^n} f(1 + \frac{k}{2^{n-3}})$$

(h)


$T_0 \leftarrow 4[f(1) + f(9)]$
$n \leftarrow 1$
$s \leftarrow 0$
$k \leftarrow 1$
$k \leftarrow k + 2$
$k \leq 2^n - 1$
F
T
$T_n = \frac{1}{2}[T_{n-1} + \frac{1}{2^{n-4}} \times s]$
$s \leftarrow s + f(1 + \frac{k}{2^{n-3}}$
$|T_n - T_{n-1}| < 10^{-3}$
T
F
$n \leftarrow n + 1$
AREA $\leftarrow T_n$

8. (a) $T_0 = \frac{1}{2}(2)[f(2) + f(4)] = [f(2) + f(4)] = (4 + 16) = 20$

$T_1 = \frac{1}{2}(1)[f(2) + 2f(3) + f(4)] = \frac{1}{2}(4 + 18 + 16) = 19$

$T_2 = \frac{1}{2}(\frac{1}{2})[f(2) + 2f(\frac{5}{2}) + 2f(3) + 2f(\frac{7}{2}) + f(4)]$

$= \frac{1}{4}(4 + \frac{25}{2} + 18 + \frac{49}{2} + 16) = \frac{75}{4}$

(b) $T_1 = \frac{1}{2}[T_0 + 2f(3)]$

$T_2 = \frac{1}{2}[T_1 + f(\frac{5}{2}) + f(\frac{7}{2})]$

(c) $T_3 = \frac{1}{2}[T_2 + \frac{1}{2}f(\frac{9}{4}) + \frac{1}{2}f(\frac{11}{4}) + \frac{1}{2}f(\frac{13}{4}) + \frac{1}{2}f(\frac{15}{4})]$

$$T_n = \frac{1}{2}T_{n-1} + \frac{2}{2^n} \sum_{\substack{k=1 \\ k \text{ odd}}}^{2^n} f(1 + \frac{2k}{2^n})$$

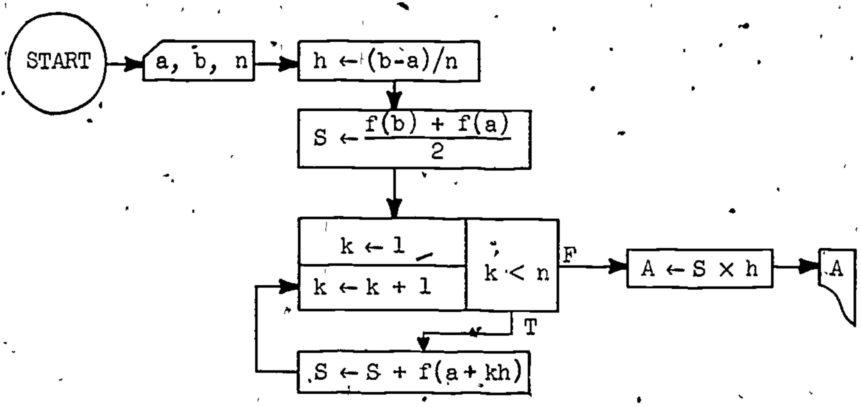
$$= \frac{1}{2}T_{n-1} + \frac{1}{2^{n-1}} \sum_{\substack{k=1 \\ k \text{ odd}}}^{2^n} f(1 + \frac{k}{2^{n-1}})$$

T7-3 The Area Under a Curve: The General Case

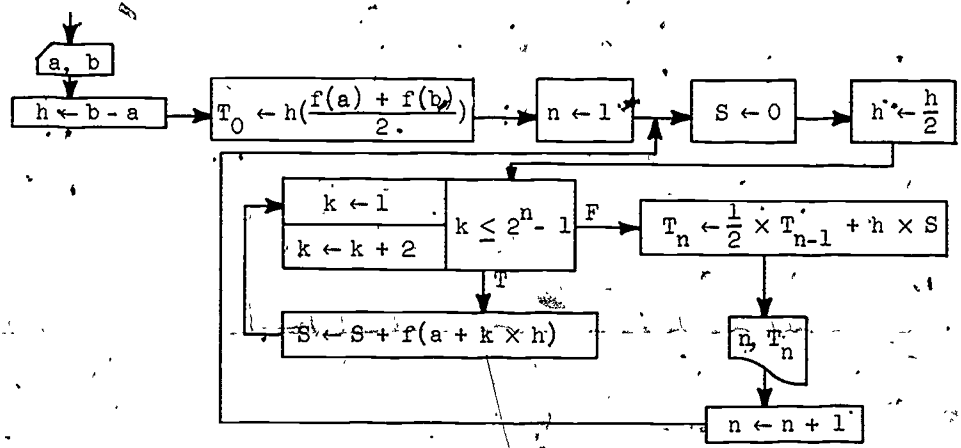
Answers to Exercises 7-3

The most difficult problems are 5(a) and 5(b). The flow chart for Problem 1 willbe called for in the language text.

1.



2.



3. We could add a tally to count the number of times the interval is cut in half and stop the calculation when this loop has been entered a certain number of times. Specifically, in box 3 we also include the statement: N ← 1. We add to box 11 the statement: N ← N + 1. Between boxes 10 and 11 we add a new decision box N ≤ N MAX. The true branch will lead to box 11 and the false to a new output box indicating an error stop.

4. Flow chart comparison.

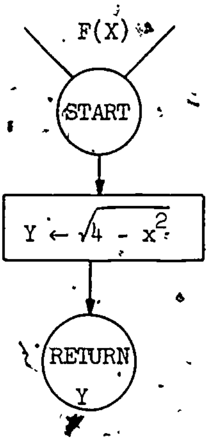
5. (a)

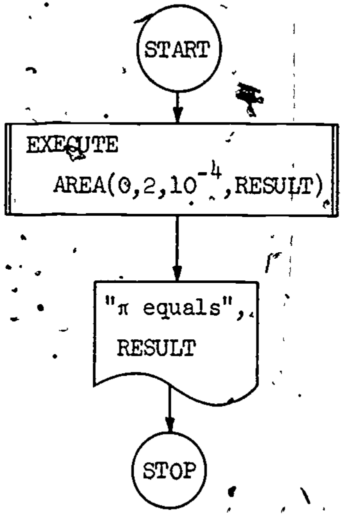
m	h	K	f(a + Kh)	S	T_1	$T_1 - T_0$
2	1	1.	9.0	9.0	19.000	10.0
4	$\frac{1}{2}$	1	6.25	6.25		
		3	12.25	18.50	18.750	0.25
8	$\frac{1}{4}$	1	5.062	5.062		
		3	7.562	12.624		
		5	10.563	23.187		
		7	14.063	37.250	18.687	0.063

(b)

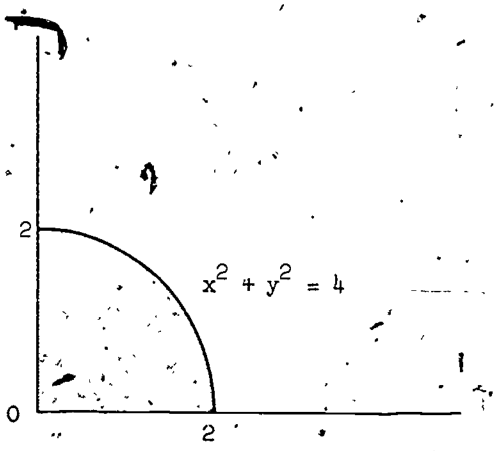
m	h	K	f(a + Kh)	S	T_1	$T_1 - T_0$
2	$\frac{3}{2}$	1	9.375	9.375	50.06	21.94
4	$\frac{3}{4}$	1	2.297	2.297		
		3	23.766	26.063	44.58	5.48
8	$\frac{3}{8}$	1	0.709	0.709		
		3	5.080	5.789		
		5	15.498	21.287		
		7	34.494	55.781	43.21	1.37
16	$\frac{3}{16}$	1	0.264	0.264		
		3	1.373	1.637		
		5	3.519	5.156		
		7	7.019	12.175		
		9	12.188	24.363		
		11	19.344	43.707		
		13	28.802	72.509		
		15	40.880	113.389	42.86	0.34

6. The easiest solution is:

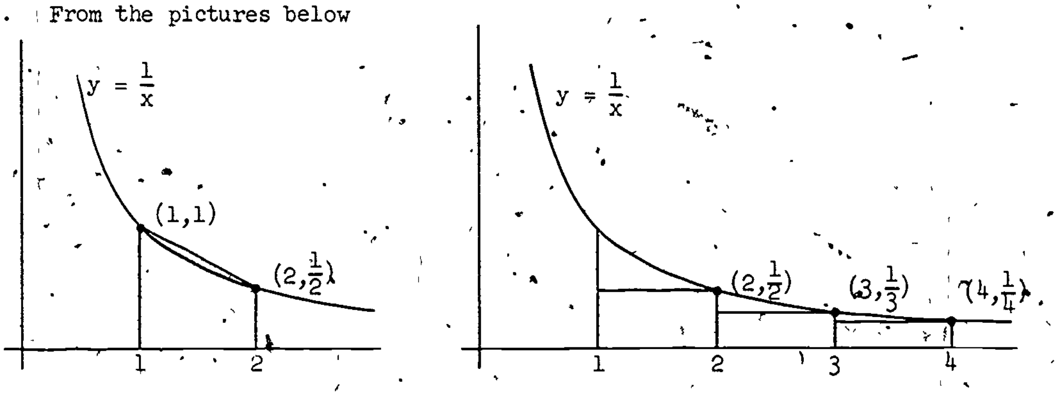




This combination of programs evaluates the area of that part of the circle $x^2 + y^2 = 4$ lying in the first quadrant and, hence, yields π.



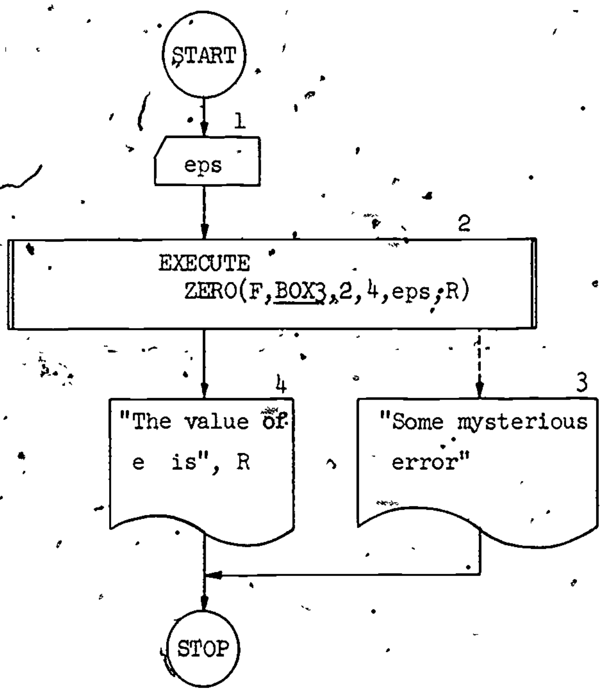
7. From the pictures below



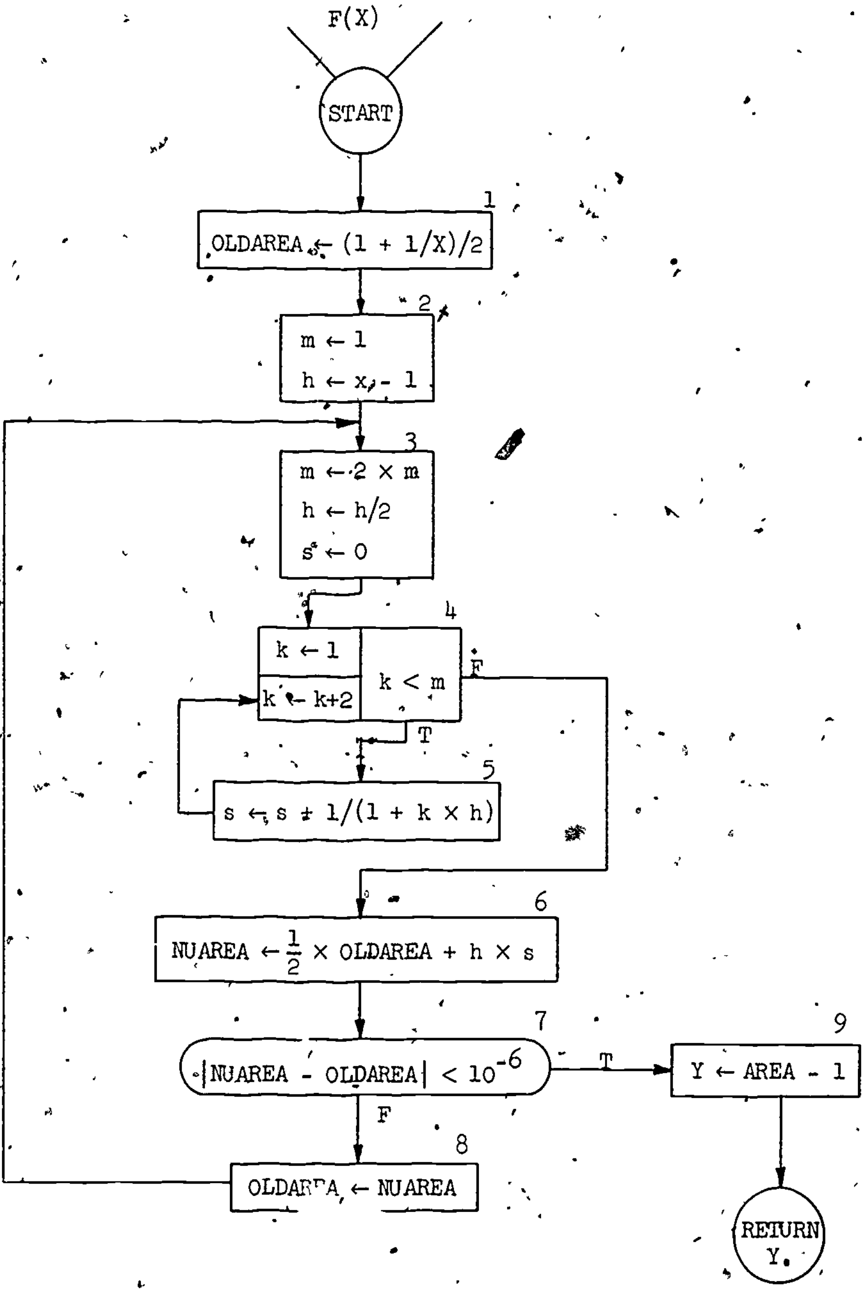
it is apparent that $\ell n2 < \frac{1 + \frac{1}{2}}{2} = \frac{3}{4}$ and $\ell n4 > \frac{1}{2} + \frac{1}{3} + \frac{1}{4} = \frac{13}{12}$. Thus,

$$F(2) = \ell n2 - 1 < -\frac{1}{4} \quad \text{and} \quad F(4) = \ell n4 - 1 > \frac{1}{12} .$$

Hence, the root of F lies between 2 and 4. Thus, our main flow chart calling on the procedure zero will be:



The flow chart for F is given next. Note that throughout the flow chart we have replaced a by 1, b by x, E by 10^{-6} and particularized f(x) at all occurrences to be 1/x. Otherwise, boxes 1 through 8 of Figure 7-20 have been unchanged. This part of the flow chart has the purpose of computing ℓnx. Finally, in box 9 we complete ℓnx - 1.



T7-4 Simultaneous Linear Equations: Developing a Systematic Method of Solution

In Sections 7-4 and 7-5 we study one of the basic problems of linear algebra, that of solving systems of linear equations. Unfortunately, it seems impossible to give the student at this level an appreciation of the wide variety of problems of pure and applied mathematics which reduce in the final analysis to the solution of systems of linear equations or to the closely related problem of matrix inversion.

The Gauss algorithm and its variants constitute the most efficient methods of calculating the solution or at least a first approximation of the solution. The familiar Cramer's Rule wherein the values of the x_j's are expressed as ratios of determinants is much less efficient for computation and is primarily of theoretical interest.

The determinant of the matrix of the coefficients (a_{ij}) can be output as a simple by-product of the improved Gaussian algorithm (Problem 2 of Exercises 7-5, Set C). This determinant is the product of all the pivot elements

$$\prod_{k=1}^{n} a_{kk}$$

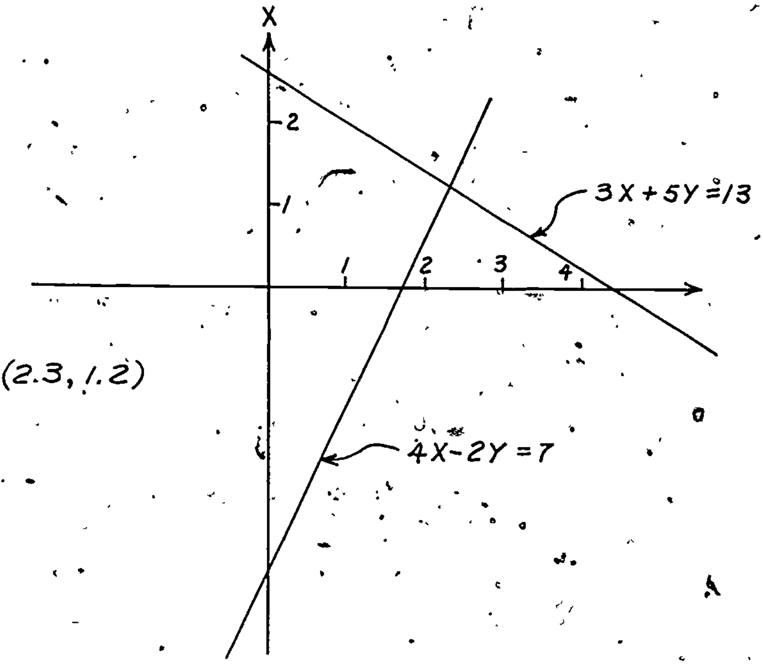
each taken after the row interchange made in the pivoting part of the algorithm. This determinant can be computed by the addition of two flow chart boxes to the algorithm. These changes are indicated in the Teacher's Commentary in connection with that problem.

The above-mentioned problem and Problem 3 of the same set (on "equilibration") will give the student a stiff workout. Experience indicates that by this time many students are thirsting for tough problems. These problems should make an excellent subject of classroom discussion once the student has had a crack at them.

Attempts are still being made to reduce the inaccuracies due to round-off error. Attempts in this direction are represented by partial pivoting and equilibration. Another method for reducing round-off error is called "complete pivoting". This method involves the permuting of variables at various stages of the process. Hence, we have the additional complication of keeping track of these permutations so as to make the inverse permutation at the end of the calculation. For this reason, and because its effectiveness is not clear, we omitted discussion of this method.

Answers to Exercises 7-4 Set A

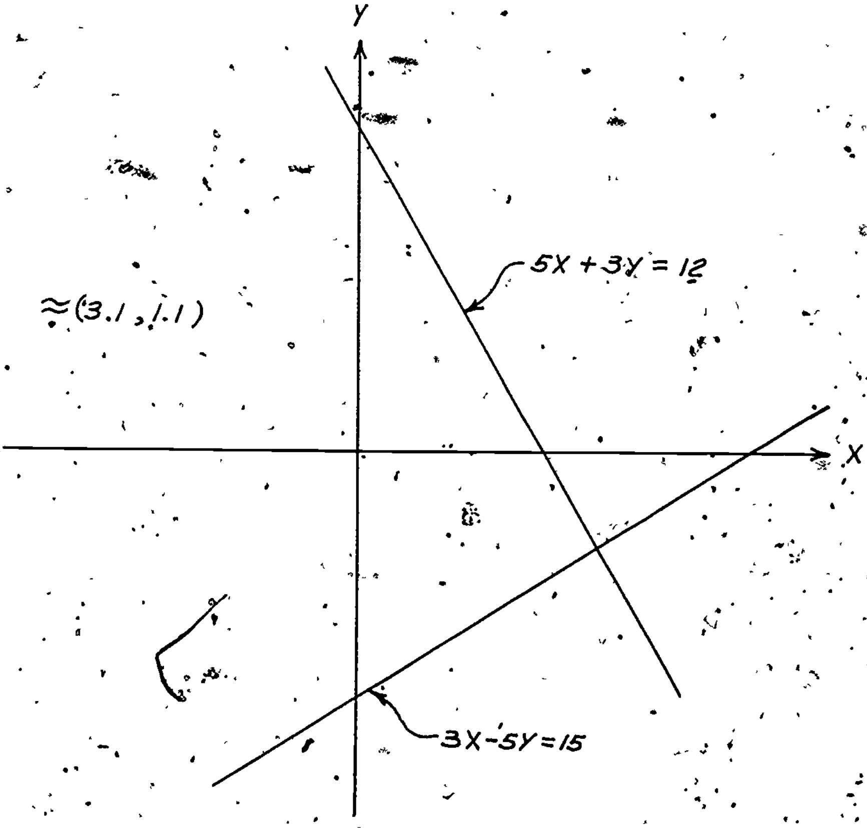
1.



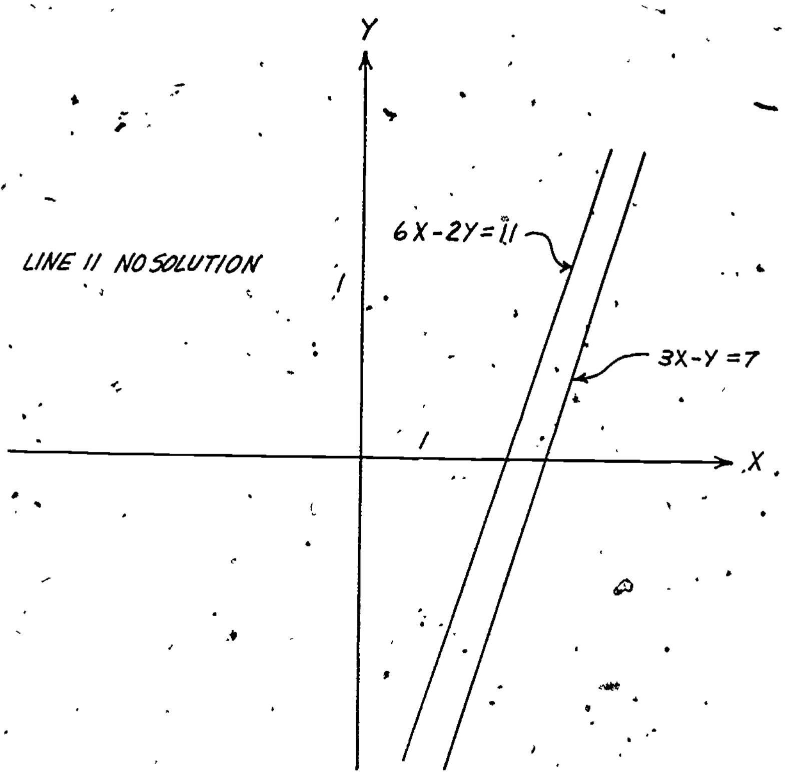
≈ (2.3, 1.2)

2.

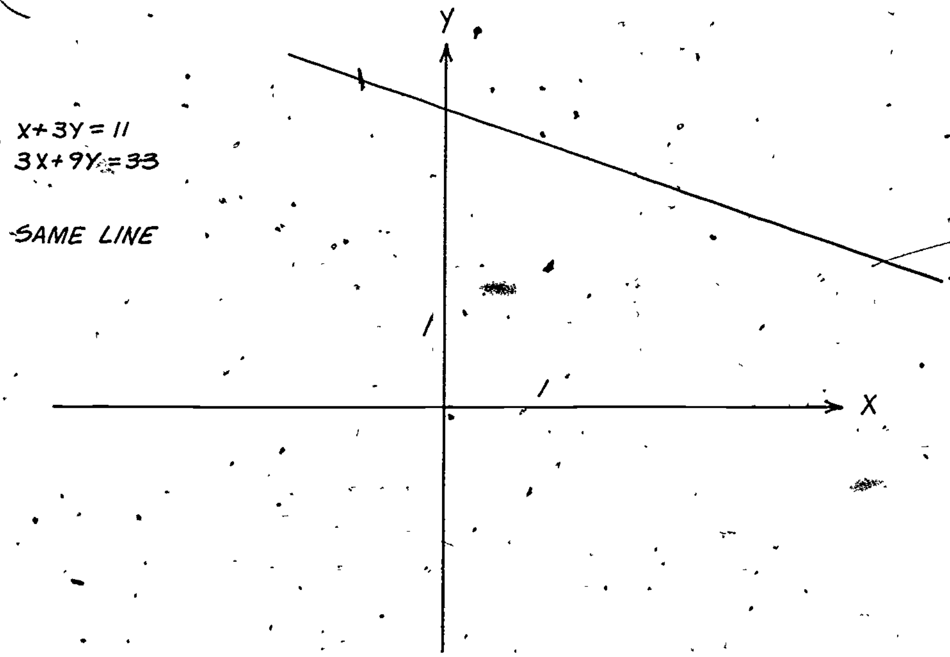
≈ (3.1, 1.1)



3.

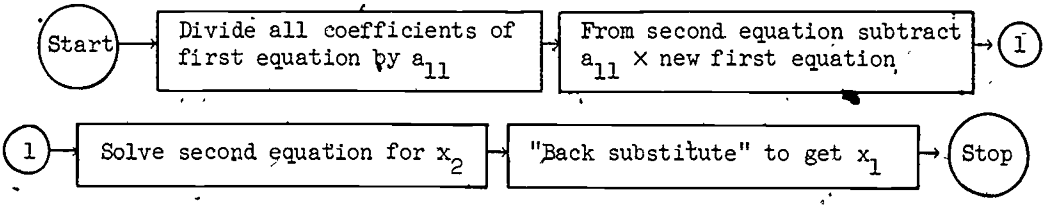

Y
X
1
1
6X-2Y=11
3X-Y=7
LINE // NO SOLUTION

4.

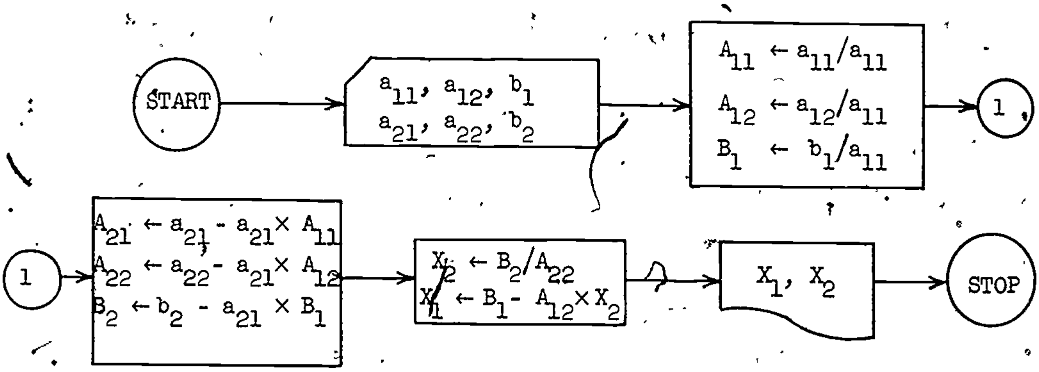

Y
X
1
1
X+3Y=11
3X+9Y=33
SAME LINE

Answer to Exercise 7-4 Set B

Assume all coefficients are different from zero.


Start
Divide all coefficients of first equation by a_{11}
From second equation subtract a_{11} × new first equation
1
1
Solve second equation for x_2
"Back substitute" to get x_1
Stop

or, more formally:


START
a_{11}, a_{12}, b_1
a_{21}, a_{22}, b_2
$A_{11} \leftarrow a_{11}/a_{11}$
$A_{12} \leftarrow a_{12}/a_{11}$
$B_1 \leftarrow b_1/a_{11}$
1
1
$A_{21} \leftarrow a_{21} - a_{21} \times A_{11}$
$A_{22} \leftarrow a_{22} - a_{21} \times A_{12}$
$B_2 \leftarrow b_2 - a_{21} \times B_1$
$X_2 \leftarrow B_2/A_{22}$
$X_1 \leftarrow B_1 - A_{12} \times X_2$
X_1, X_2
STOP

The next section of the text shows how to solve equations more efficiently.

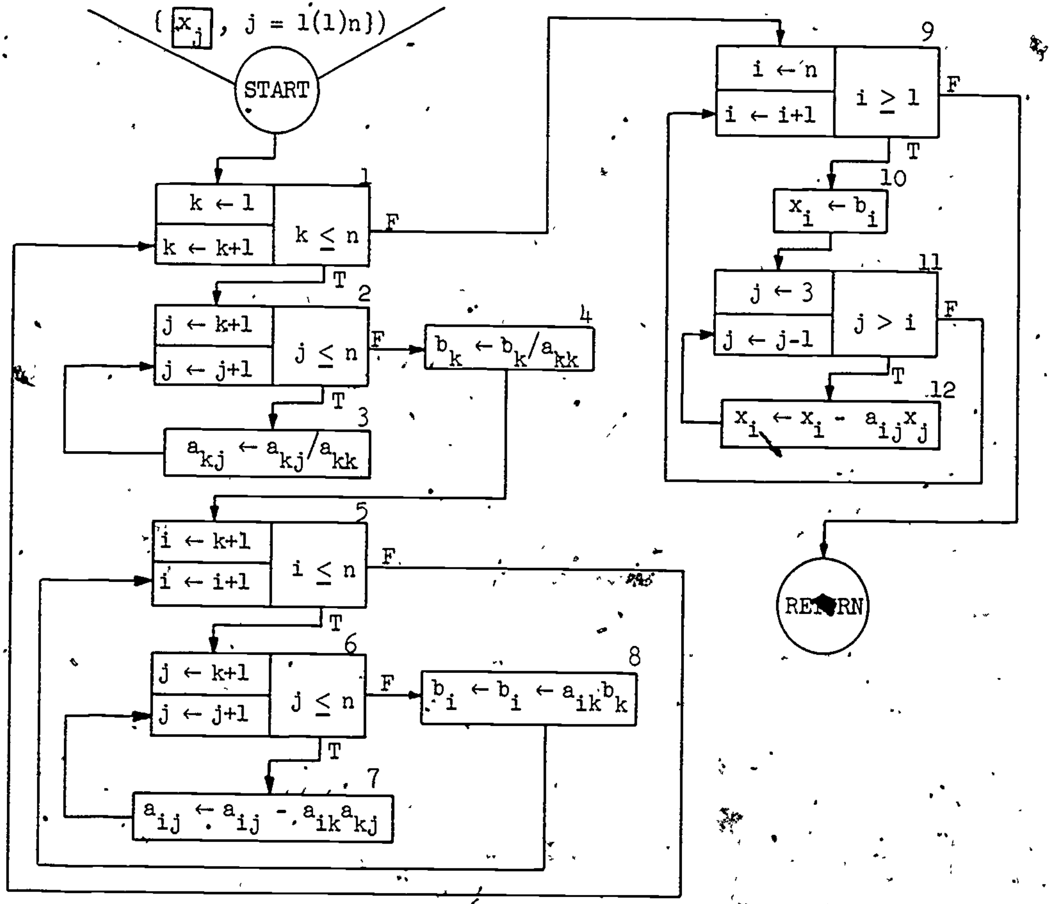
Answers to Exercises 7-5 Set A

1. Upper bounds shown as $\underline{\underline{3}}$ murt be changed to n. These changes occur in boxes 0, 1, 2, 5, 6 and 13. Also, the initial values for control indexes in bcxes 9 and 11 should be changed from 3 to n.

2. GAUSS(n, $\{\{\underset{\sim}{a_{ij}}, j = 1(1)n\} i = 1(1)n\}$,

$\{\underset{\sim}{b_i}, i = 1(1)n\}$

$\{\boxed{x_j}, j = 1(1)n\}$)



Gaussian algorithm for n equations in n unknowns

(without pivoting)

Answers to Exercises 7-5 Set B

2. The sequence of arrays is

$$\begin{bmatrix} 3 & 4 & 1 & -7 \\ 2 & 4 & 1 & 3 \\ 3 & -5 & 3 & 7 \end{bmatrix} \quad \begin{bmatrix} \boxed{3} & \frac{4}{3} & \frac{1}{3} & -\frac{7}{3} \\ 2 & 4 & 1 & 3 \\ 3 & -5 & 3 & 7 \end{bmatrix} \quad \begin{bmatrix} \boxed{3} & \frac{4}{3} & \frac{1}{3} & -\frac{7}{3} \\ \textcircled{2} & \frac{4}{3} & \frac{1}{3} & \frac{23}{3} \\ \textcircled{3} & -9 & 2 & 14 \end{bmatrix}$$

$$\begin{bmatrix} \boxed{3} & \frac{4}{3} & \frac{1}{3} & -\frac{7}{3} \\ \textcircled{2} & \boxed{\frac{4}{3}} & \frac{1}{4} & \frac{23}{4} \\ \textcircled{3} & -9 & 2 & 14 \end{bmatrix} \quad \begin{bmatrix} \boxed{3} & \frac{4}{3} & \frac{1}{3} & -\frac{7}{3} \\ \textcircled{2} & \boxed{\frac{4}{3}} & \frac{1}{4} & \frac{23}{4} \\ \textcircled{3} & \textcircled{-9} & \frac{17}{4} & \frac{263}{4} \end{bmatrix} \quad \begin{bmatrix} \boxed{3} & \frac{4}{3} & \frac{1}{3} & -\frac{7}{3} \\ \textcircled{2} & \boxed{\frac{4}{3}} & \frac{1}{4} & \frac{23}{4} \\ \textcircled{3} & \textcircled{-9} & \boxed{\frac{17}{4}} & \frac{263}{17} \end{bmatrix}$$

For the back solutions, $x_3 = \frac{263}{17}$ and x_2 and x_1 successively receive the values indicated

$$x_2 = \frac{23}{4},\ \frac{23}{4} - \frac{1}{4} \cdot \frac{263}{17} = \frac{32}{17}$$

$$x_1 = -\frac{7}{3},\ -\frac{7}{3} - \frac{1}{3} \cdot \frac{263}{17} = -\frac{382}{17},\ -\frac{382}{17} - \frac{4}{3} \cdot \frac{32}{17} = 10\ .$$

T7

8. (a) $$\begin{bmatrix} 3 & -2 & 7 & -1 & 2 \\ 2 & 3 & -4 & 1 & 7 \\ 1 & 2 & 5 & 2 & 11 \\ 4 & 3 & 7 & -8 & -2 \end{bmatrix}$$

$$\begin{bmatrix} \boxed{3} & -0.667 & 2.33 & -0.333 & 0.667 \\ 2 & 3 & -4 & 1 & 7 \\ 1 & 2 & 5 & 2 & 11 \\ 4 & 3 & 7 & -8 & -2 \end{bmatrix}$$

$$\begin{bmatrix} \boxed{3} & -0.667 & 2.33 & -0.333 & 0.667 \\ \textcircled{2} & 4.33 & -8.67 & 1.67 & 5.67 \\ \textcircled{1} & 2.67 & 2.67 & 2.33 & 10.3 \\ \textcircled{4} & 5.67 & -2.33 & -6.67 & -4.67 \end{bmatrix}$$

$$\begin{bmatrix} \boxed{3} & -0.667 & 2.33 & -0.333 & 0.667 \\ \textcircled{2} & \boxed{4.33} & -2.00 & 0.385 & 1.31 \\ \textcircled{1} & 2.67 & 2.67 & 2.33 & 10.3 \\ \textcircled{4} & 5.67 & -2.33 & -6.67 & -4.67 \end{bmatrix}$$

$$\begin{bmatrix} \boxed{3} & -0.667 & 2.33 & -0.333 & 0.667 \\ \textcircled{2} & \boxed{4.33} & -2 & 0.385 & 1.31 \\ \textcircled{1} & \textcircled{2.67} & 8.00 & 1.302 & 6.802 \\ \textcircled{4} & \textcircled{5.67} & 9.010 & -8.853 & -12.098 \end{bmatrix}$$

$$\begin{bmatrix} \boxed{3} & -0.667 & 2.33 & -0.333 & 0.667 \\ \textcircled{2} & \boxed{4.33} & -2 & 0.385 & 1.31 \\ \textcircled{1} & \textcircled{2.6} & \boxed{8} & 0.163 & 0.856 \\ \textcircled{4} & \textcircled{5.67} & 9.00 & -8.85 & -12.1 \end{bmatrix}$$

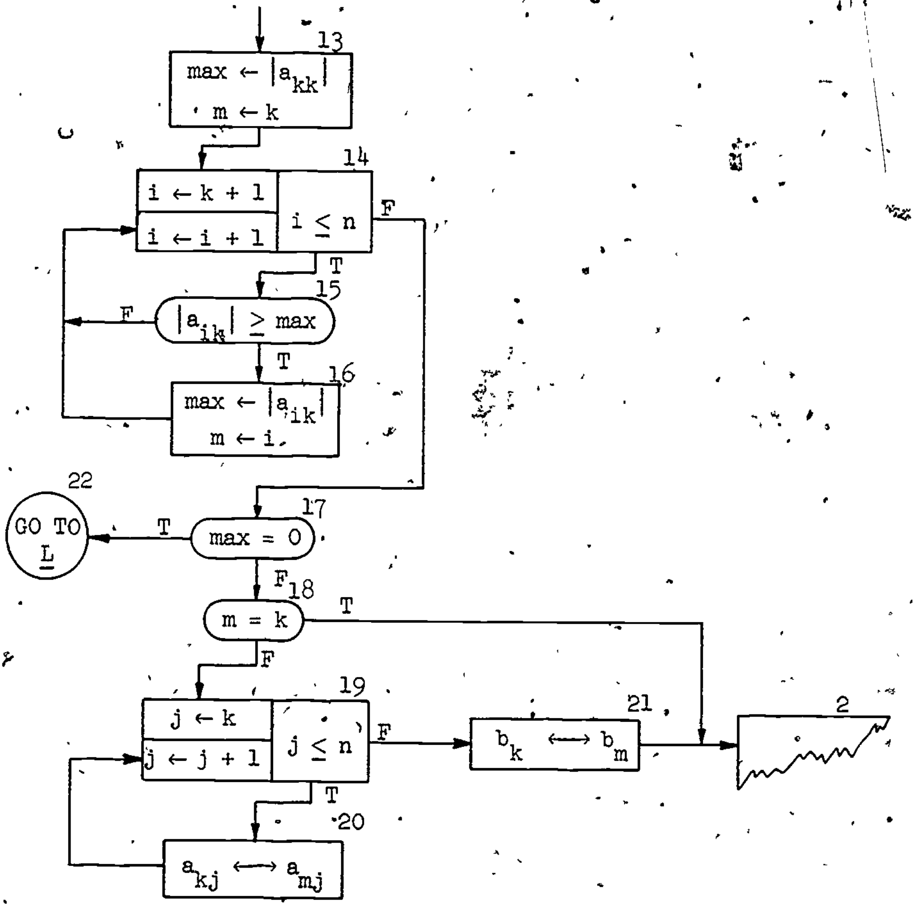
$$\begin{bmatrix} \boxed{3} & -0.667 & 2.33 & -0.333 & 0.667 \\ \textcircled{2} & \boxed{4.33} & -2 & 0.385 & 1.31 \\ \textcircled{1} & \textcircled{2.67} & \boxed{8} & 0.163 & 0.856 \\ \textcircled{4} & \textcircled{5.67} & \textcircled{9.00} & -10.3 & -19.8 \end{bmatrix}$$

$$\begin{bmatrix} \boxed{3} & -0.667 & 2.33 & -0.333 & 0.667 \\ \textcircled{2} & \boxed{4.33} & -2 & 0.385 & 1.31 \\ \textcircled{1} & \textcircled{2.67} & \boxed{8} & 0.163 & 0.856 \\ \textcircled{4} & \textcircled{5.67} & \textcircled{9.00} & \boxed{-10.3} & 1.917 \end{bmatrix}$$

$$\begin{aligned} x_1 - 0.667x_2 + 2.33x_3 - 0.333x_4 &\doteq 0.667 \\ x_2 - 2x_3 + 0.385x_4 &\doteq 1.31 \\ x_3 + 0.163x_4 &\doteq 0.856 \\ x_4 &\doteq 1.92 \end{aligned}$$

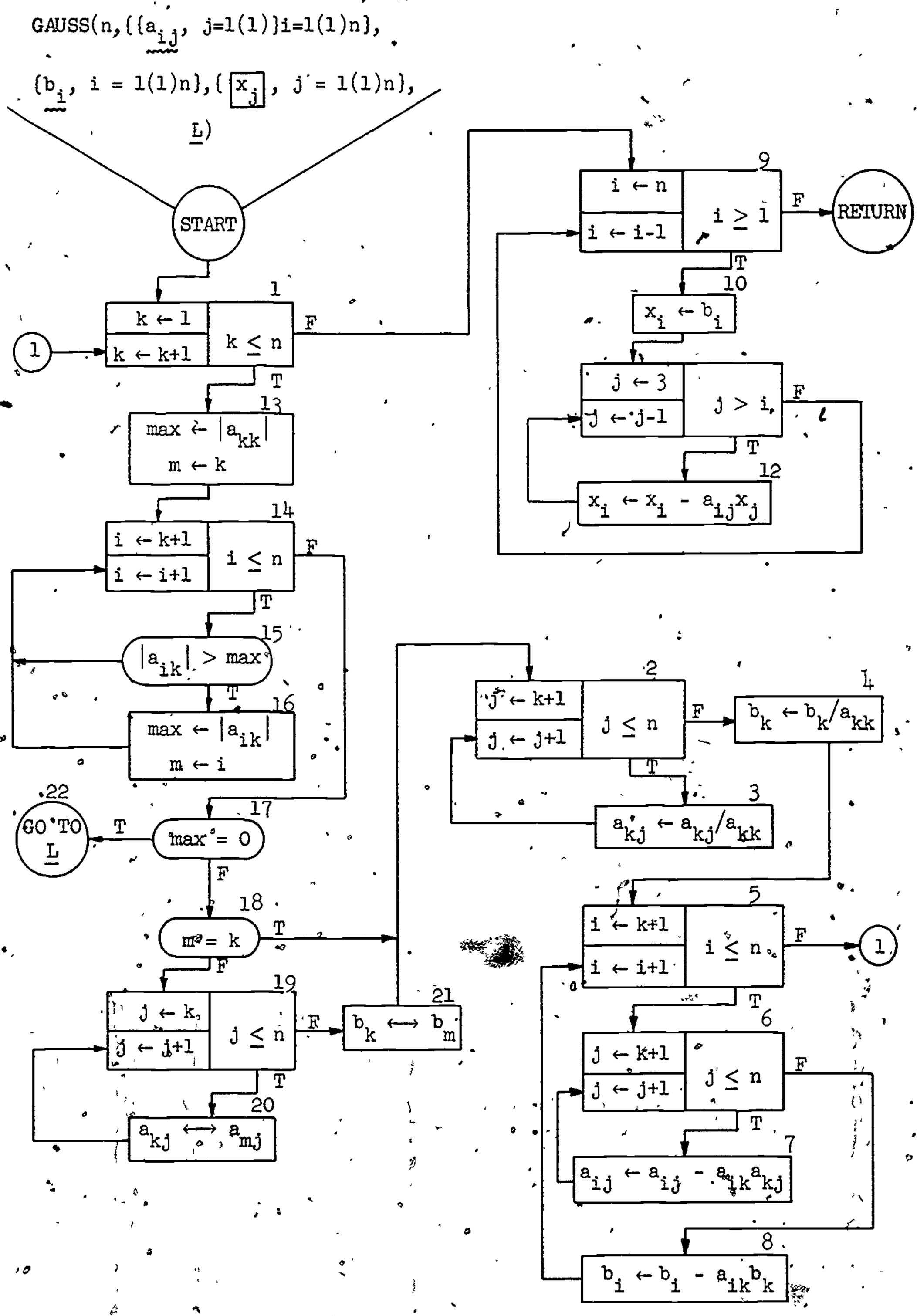
Answers to Exercises 7-5 Set C

1.



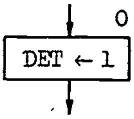
Flow chart fragment for partial pivoting

T7

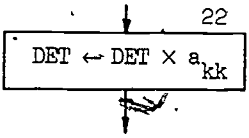


Gaussian algorithm for n equations in n unknowns with partial pivoting

In order to have this flow chart also compute the determinant of the matrix, (a_{ij}), add two flow chart boxes as follows. Add

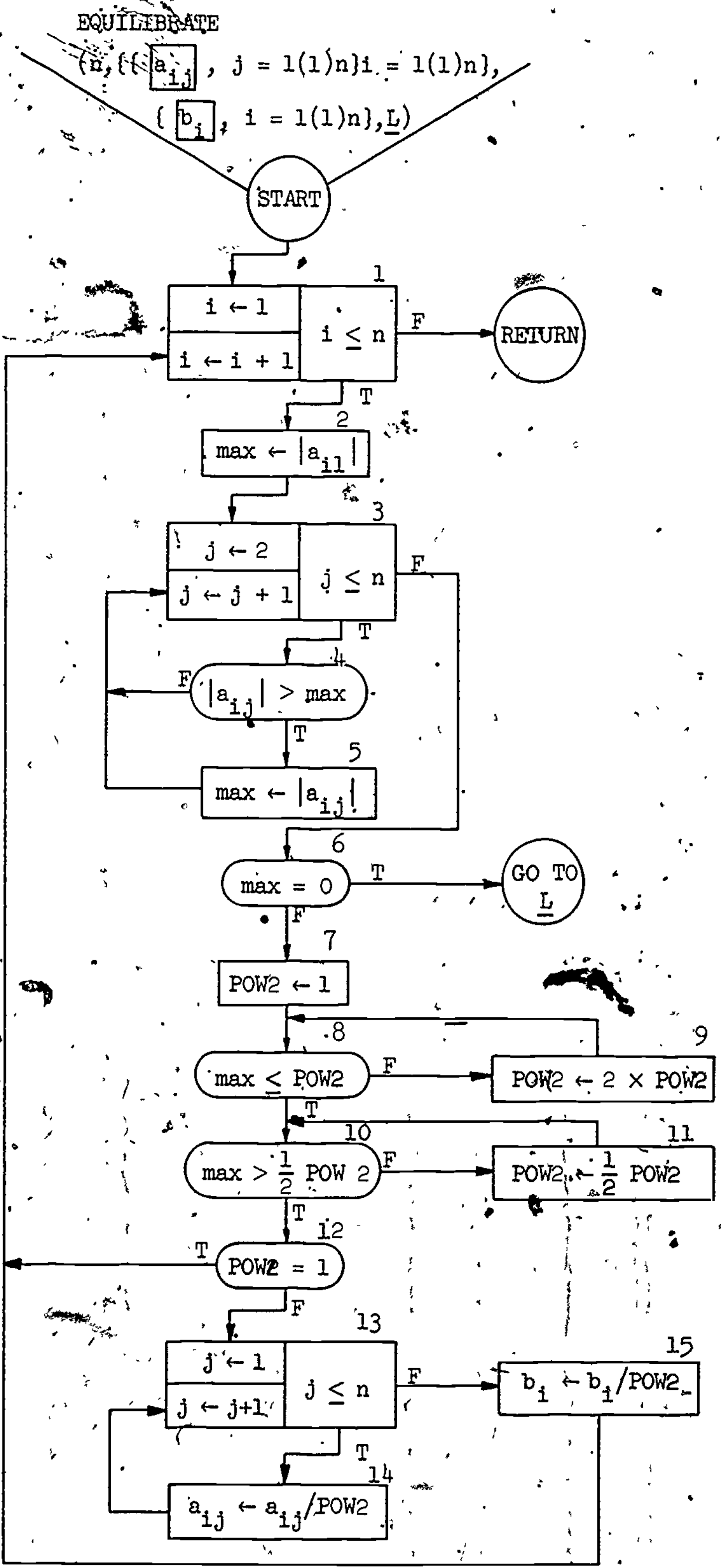


immediately before box 1. Add

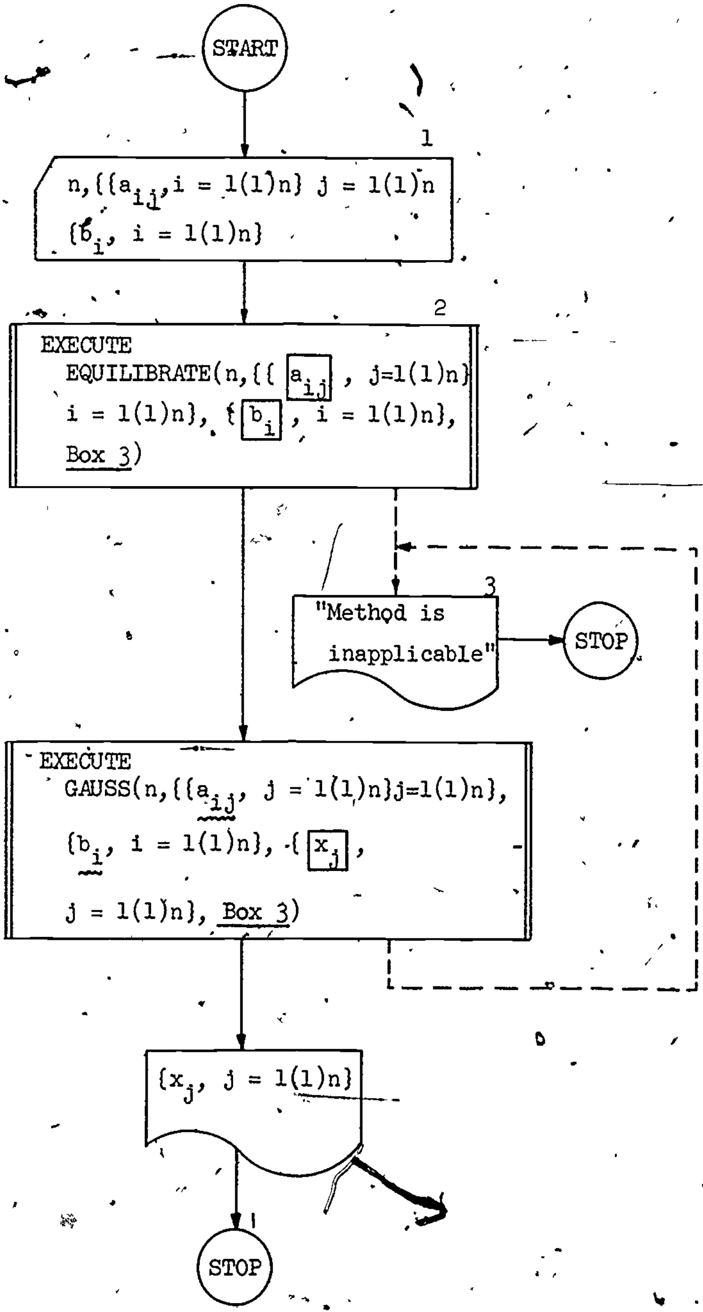


immediately before box 2. The final value of DET will be the determinant of the (a_{ij}). A receptacle must also be supplied in the hopper for returning this value to the main flow chart.

2.



Equilibration algorithm


START
1
n, {{a_{ij}, i = 1(1)n} j = 1(1)n
{b_i, i = 1(1)n}
2
EXECUTE
EQUILIBRATE(n, {{ $\boxed{a_{ij}}$, j=1(1)n}
i = 1(1)n}, {$\boxed{b_i}$, i = 1(1)n},
Box 3)
3
"Method is inapplicable"
STOP
EXECUTE
GAUSS(n, {{a_{ij}, j = 1(1)n}j=1(1)n},
{b_i, i = 1(1)n}, {$\boxed{x_j}$,
j = 1(1)n}, Box 3)
{x_j, j = 1(1)n}
STOP

Call for solution of system of equations with equilibration

Sample Test Questions

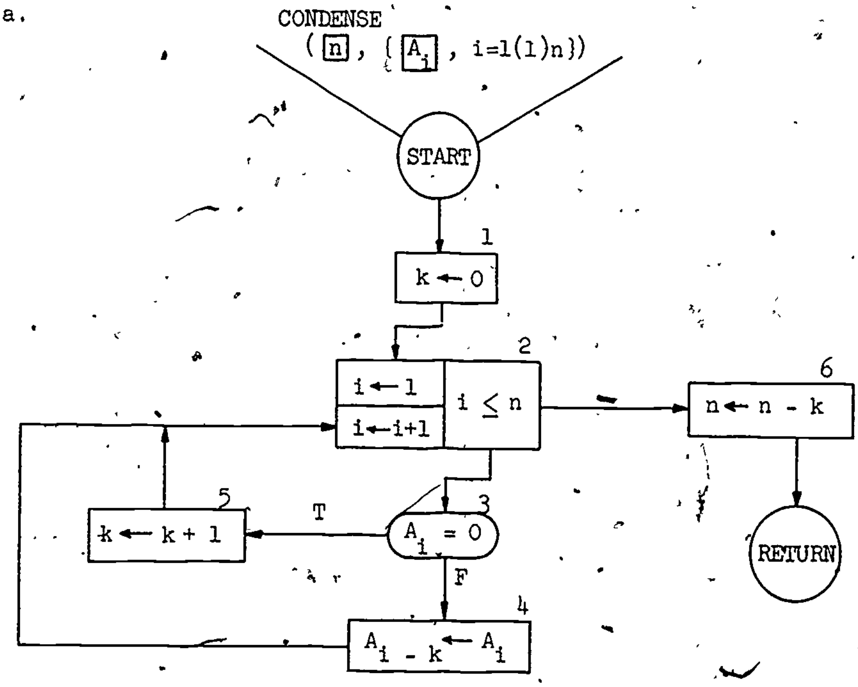
7. (Chapter 5)

a. Prepare a reference procedure flow chart, CONDENSE, which receives a vector $A_1, A_2, \ldots, A_n$ and returns in consecutive positions the non-zero elements of the original vector and also returns as n, the dimension of the new vector. (Don't worry about the values of A_i with i greater than the new n.)

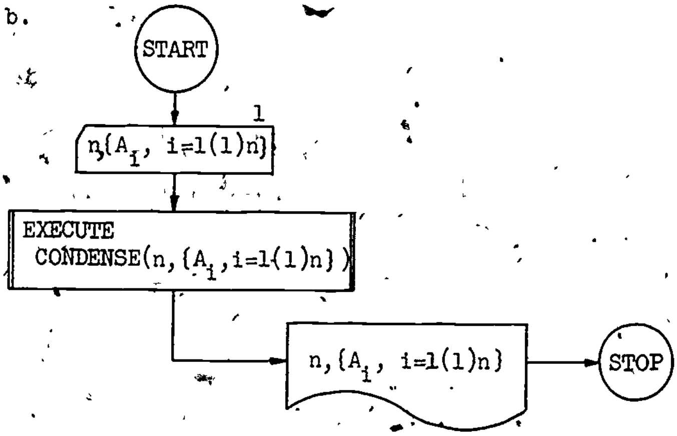
b. Prepare a flow chart to read in a vector X, use CONDENSE, and print out the condensed vector and its dimension.

Solution:

a.



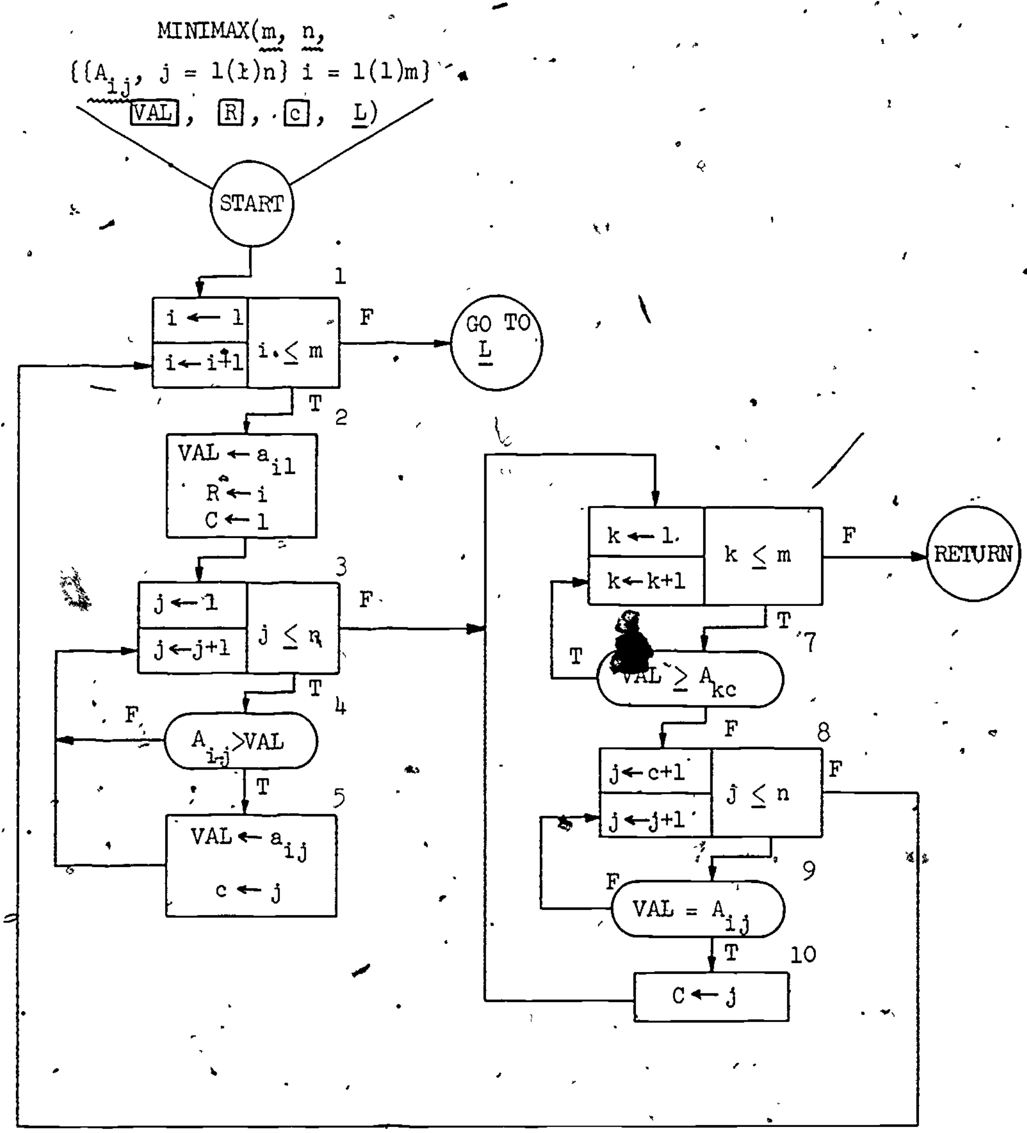
b.



8. (Chapter 5)

Write a procedure flow chart MINIMAX for inputting an m by n matrix and searching it for an element which is at the same time as large as any element in its row and as small as any element in its column. Return the location and value of this element. Provide a special exit for the case that there is no such element.

Solution:



[Note to teacher: If the student were allowed to assume that no two elements in any row had the same value, then the loop in Boxes 8 - 10 could be eliminated and the F exit from Box 7 hooked into the incrementation section of Box 1. The algorithm would thus be considerably simplified.]

Take Home Exam Question (Chapter 7)

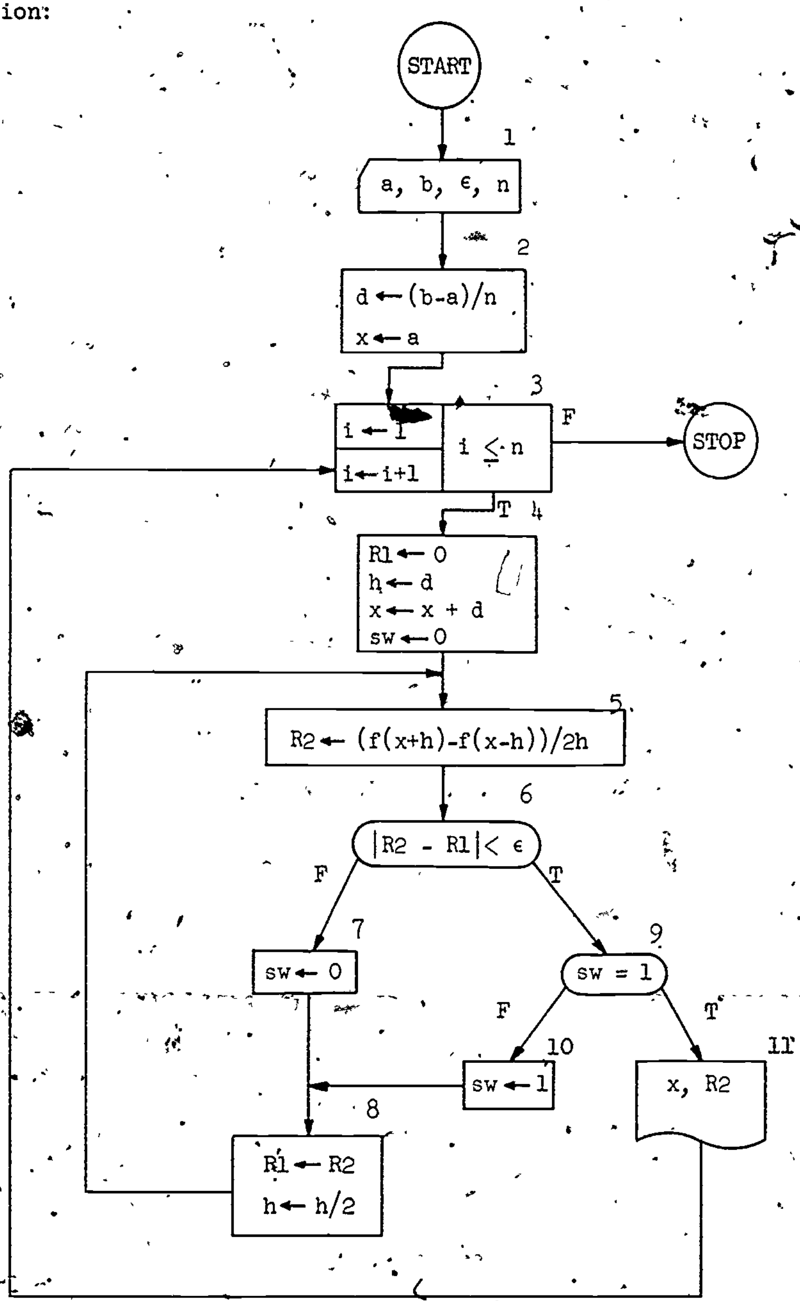
It is often important in mathematics to know the rate of change of a function at various points. The rate of change of a function on an interval is defined as the difference of the functional values at the endpoints of the interval divided by the length of the interval. For example:

$$\frac{f(x + h) - f(x - h)}{2h}$$

is the rate of change of f over the interval $[x - h, x + h]$. If h is so small that reduction of the size of h makes no perceptible change in the computed rate, we will take the computed rate as the rate of change of the function at x.

Your job is to prepare a flow chart computing the rate of change of a reference function, f, at $n - 1$ points, which divide an input interval $[a,b]$ into n equal subintervals. Successively halve the values of h, initializing h with the length of the subintervals. Output the value of the rate when it remains stationary (differences less than an input value of ϵ) for three consecutive trials.

Solution:



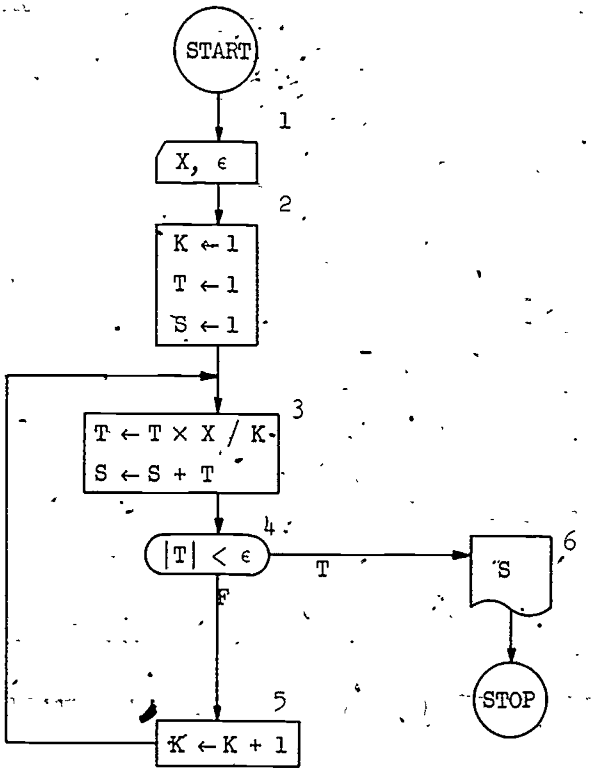
START
1
a, b, ϵ, n
2
d ← (b-a)/n
x ← a
3
i ← 1
i ← i+1
i ≤ n
F
STOP
T
4
R1 ← 0
h ← d
x ← x + d
sw ← 0
5
R2 ← (f(x+h)-f(x-h))/2h
6
|R2 - R1| < ϵ
F
T
7
sw ← 0
9
sw = 1
F
T
10
sw ← 1
11
x, R2
8
R1 ← R2
h ← h/2

9. (Chapter 6) An important function in later math courses is defined by

$$EXP(X) = 1 + X + \frac{X^2}{1 \cdot 2} + \frac{X^3}{1 \cdot 2 \cdot 3} + \frac{X^4}{1 \cdot 2 \cdot 3 \cdot 4} + \cdots .$$

Construct a flow chart for inputting X and ϵ and computing approximations to EXP(X). Terminate the process and print out the result when two consecutive approximations differ ~~by~~ less than ϵ.

Solution:



10. (Chapter 7) Hand trace through the flow chart ~~of~~ Figure 7-20 with the function $f(x) = 3x + 5$ and with the input values of a, b, ϵ respectively 1, 4 and .01. List in the table below the values successively assigned the indicated variables. Encircle the output value of NUAREA.

m	h	s	OLDAREA	NUAREA

Answer the following questions concerning this trace.

a) How many times was the test in Box 7 made?

b) How many times was Box 8 executed?

c) What property of the particular function caused the process to terminate when it did.

Solution:

m	h	s	OLDAREA	NUAREA
1	3	0	75/2	75/2
2	3/2	25/2		

a) Once

b) None

c) Since the graph of the function is a straight line the area to be approximated is a trapezoid and all trapezoidal "approximations" will be exact. Thus the first two approximations have a difference of 0 which is less than .01.

Chapter T8

COMPILATION AND SOME OTHER NON-NUMERIC PROBLEMS

8-1 Introduction

This chapter is expected to be fairly difficult. Not only is the material somewhat different from most of the material in the course but also the chapter deals with problems of somewhat greater complexity than those found earlier in the text. Many classes may not be able to cover this material in detail. If desirable due to lack of time, Section 8-4 may be skipped; but, if it is skipped, the results of the processes presented there should be summarized for the students. These results are:

1. Program statements are read from cards. Individual statements are separated from each other and statements which may run from one card to the next are properly joined together.
2. All blank spaces are eliminated from each statement.
3. Each statement is identified as to type, and
4. Internal symbols are substituted for all symbols and symbol strings used by the program to identify variables, so that every symbol is just one element of the resulting string.

When these results are understood, the student can proceed to the more exciting results of Sections 8-5 and 8-6.

8-2 Symbol Manipulation

In this section, the manipulation of symbols is motivated by the possibility of determining authorship of literature when the identity of the true author is in doubt. You might refer to the article by Frederick Mostellar in the American Statistician (1963) where the authorship of the Federalist papers was determined (mostly by Hamilton) with a high degree of probability. The class should not be led to think that authorship can always (or easily) be determined. A number of other attempts have led to highly questionable results.

Answers to Exercises 8-2

1. Determine if the character 'A' occurs at any place after a 'B'. If so, return a pointer to 'A' (set p = the index of 'A').

To solve this problem, we must search the string twice, once for 'B' and then for 'A'. The first search begins with the first character and the second at the character just beyond the 'B', if the 'B' is present. A flow chart solution is:

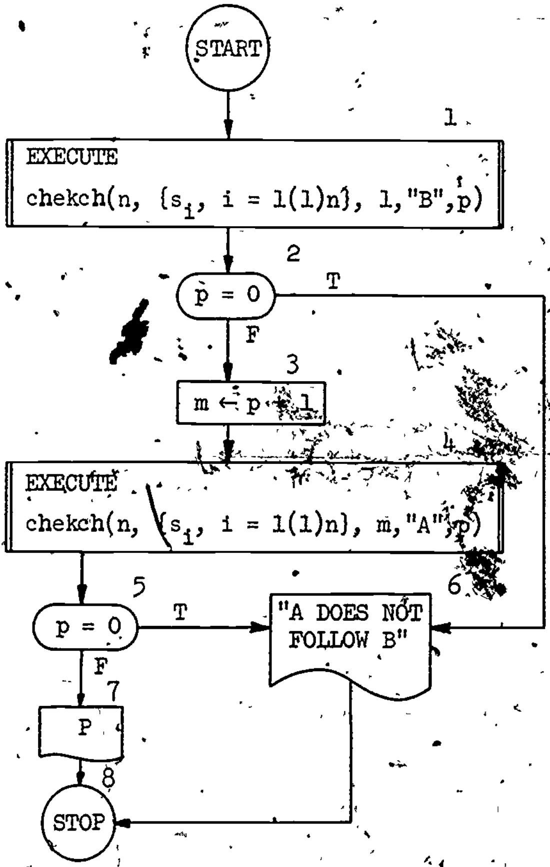


Figure T8-2

2. Determine if the substring 'TH' occurs at any place after the substring 'DR'. If so, return a pointer to 'DR'.

This problem is similar to No. 1, except that now substrings are sought, so chekst must be used. Also, a pointer is to be returned to the first substring, rather than the second.

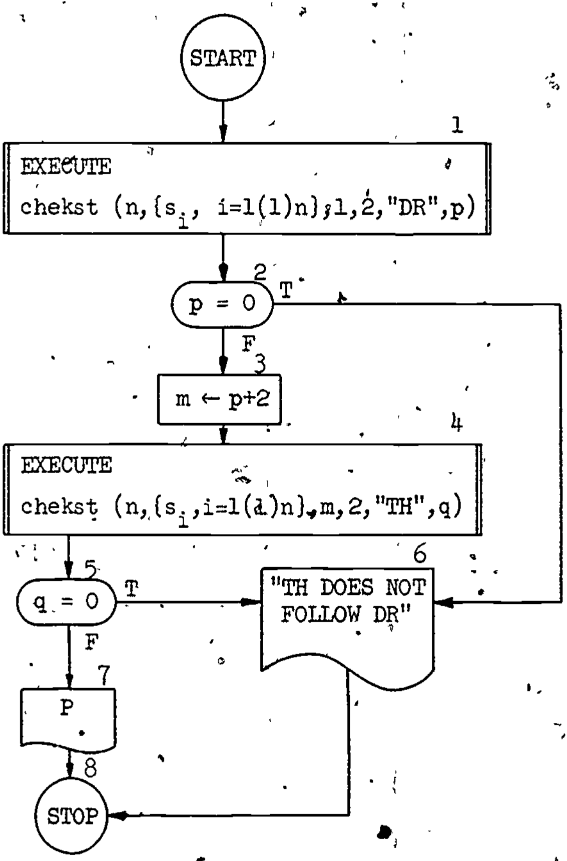


Figure T8-3

3. Determine if the characters 'A', 'B', and 'C' occur in that order, not necessarily adjacently. If so, return a pointer to 'B'.

This problem is similar to No. 1 and No. 2. Now, three searches must be made. Since we must return a pointer to 'B' (if conditions are met), we need a new pointer (q) for the search for 'C'.

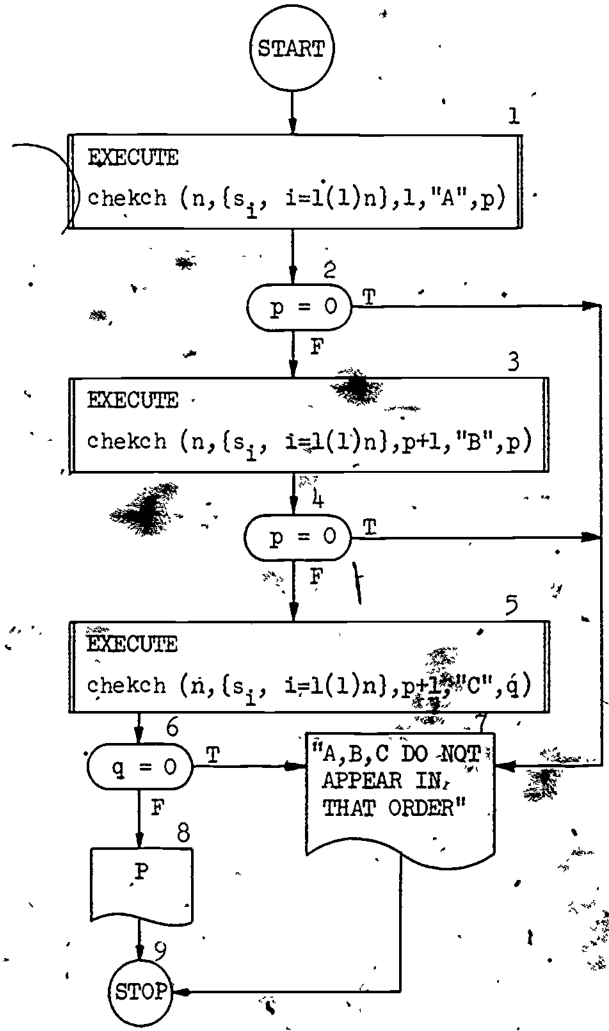

START
1
EXECUTE
chekch (n,{s_i, i=1(1)n},1,"A",p)
2
p = 0
T
F
3
EXECUTE
chekch (n,{s_i, i=1(1)n},p+1,"B",p)
4
p = 0
T
F
5
EXECUTE
chekch (n,{s_i, i=1(1)n},p+1,"C",q)
6
q = 0
T
F
7
"A,B,C DO NOT APPEAR IN THAT ORDER"
8
P
9
STOP

Figure T8-4

4. Identify the most frequently occurring character in the string.

A search for 26 letters is required, with a count of each resulting. For this purpose, the contch procedure is used. Let the set of letters be ℓ_1, i = 1, ..., 26. Let L be the most common letter and N its count; these two values are to be printed by the program.

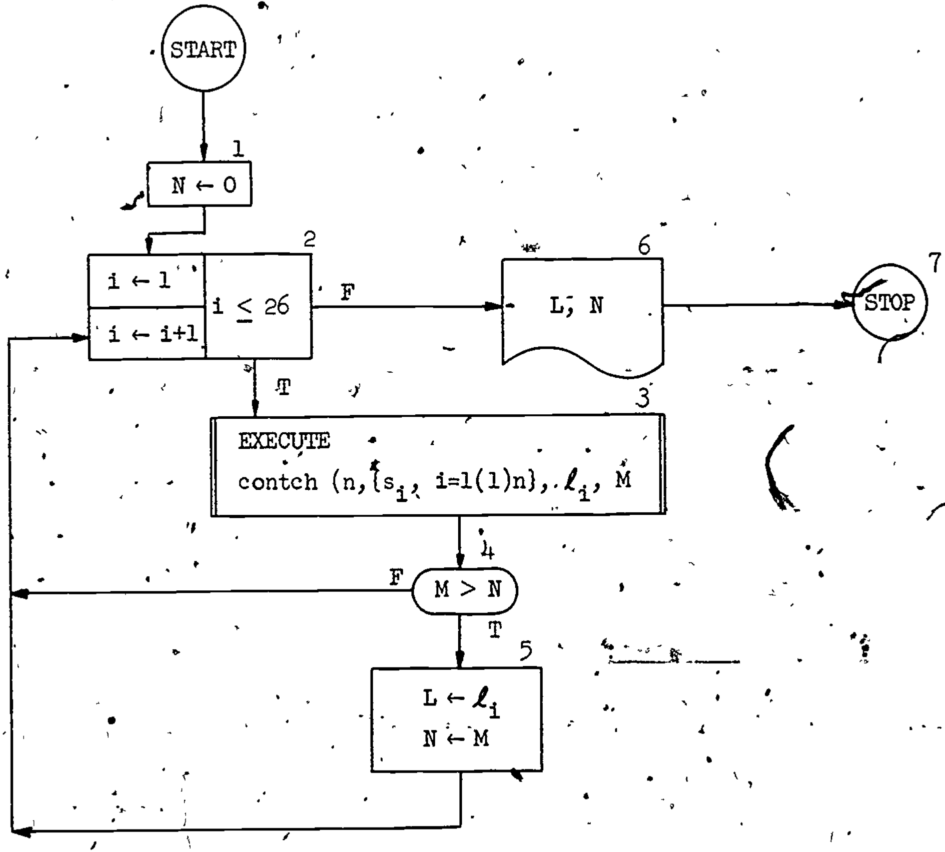


Figure T8-5

5. Find out if any letters of the alphabet occur exactly three times and identify them.

The contch procedure is useful here also. After it is executed for each letter, we check the count. If the count is 3, we print the letter. (Alternately, if we wished to have such letters printed at the end of the search, we could save them and print them later.)

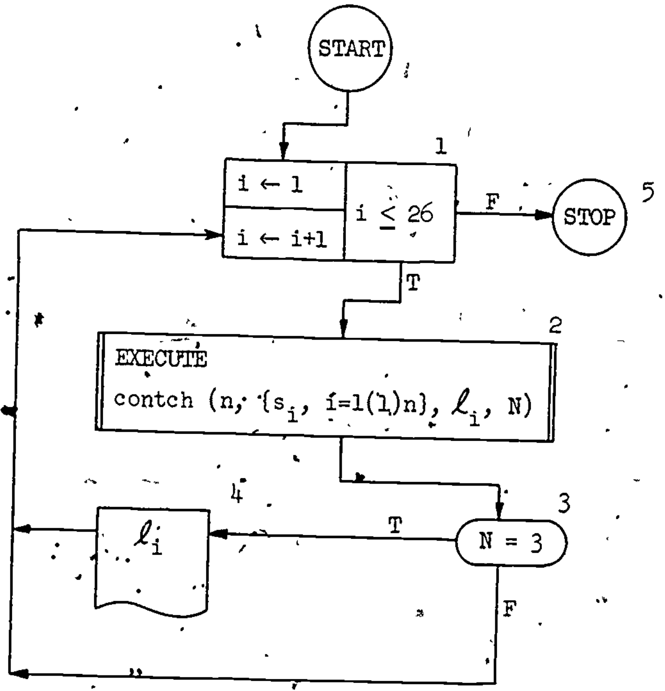


Figure T8-6

6. If 'B' immediately follows 'A', substitute 'X' for each such 'B' in the string. Report the count of such substitutions.

We can state the problem this way: for each substring 'AB', replace the 'B' by 'X', and count substitutions. We call repeatedly on chekst to locate appearances of 'AB' and substitute 'X' for the character after the one to which p points. (In Problems 1 - 3 we needed multiple searches because characters there could be non-adjacent. Here we seek adjacent 'A' and 'B' and so one use of chekst suffices.)

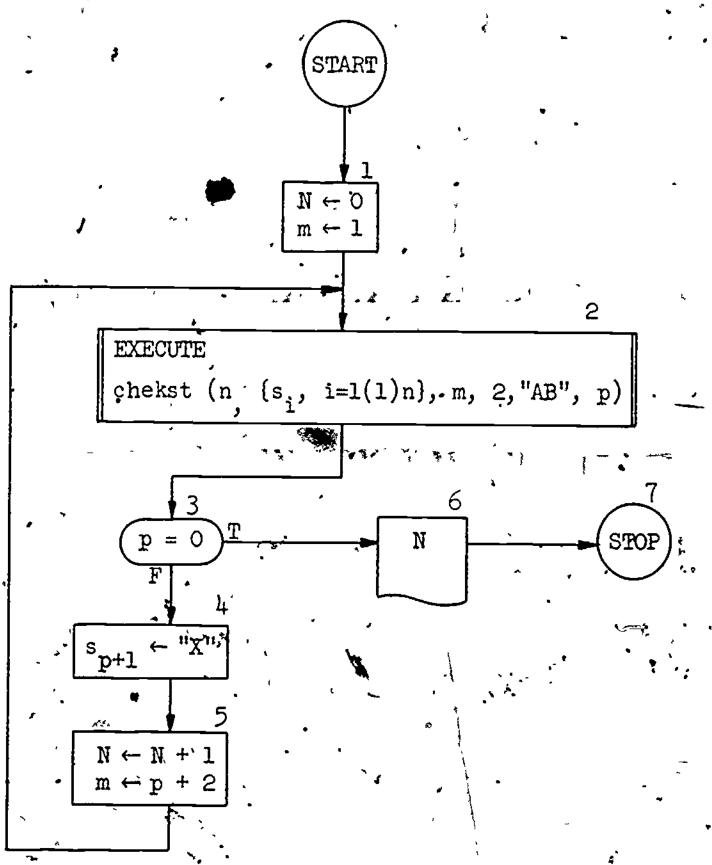
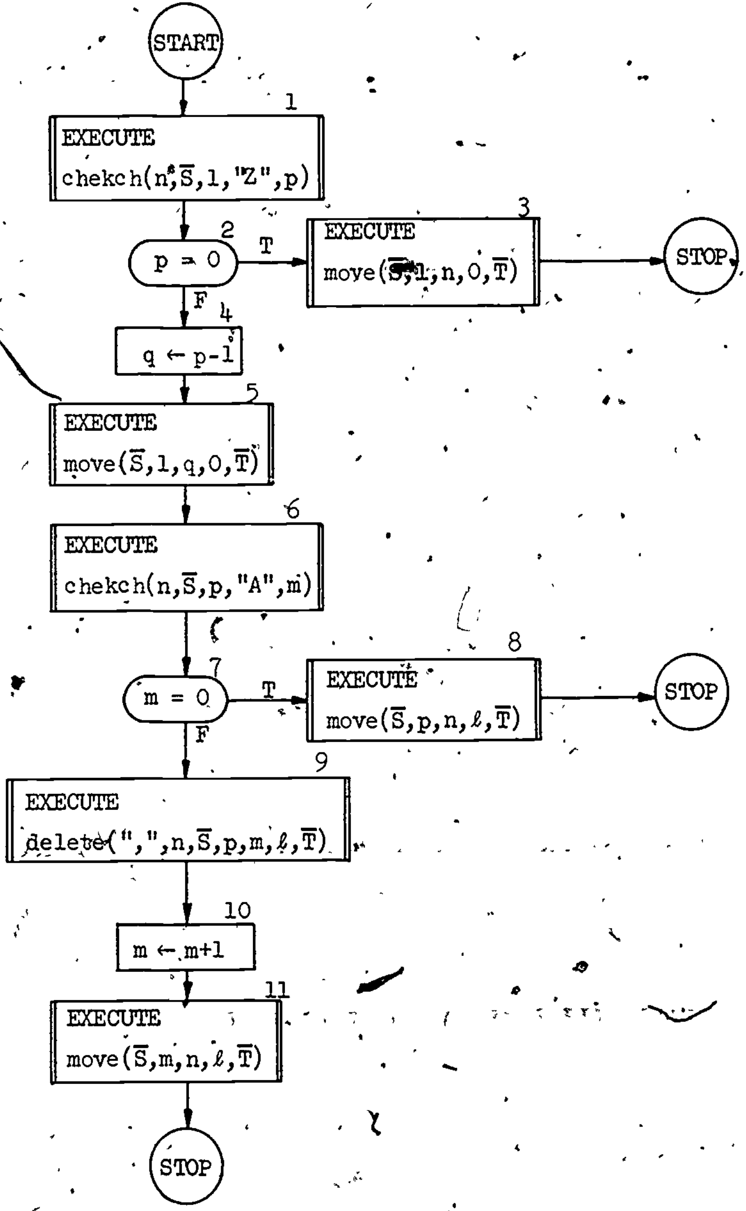


Figure T8-7

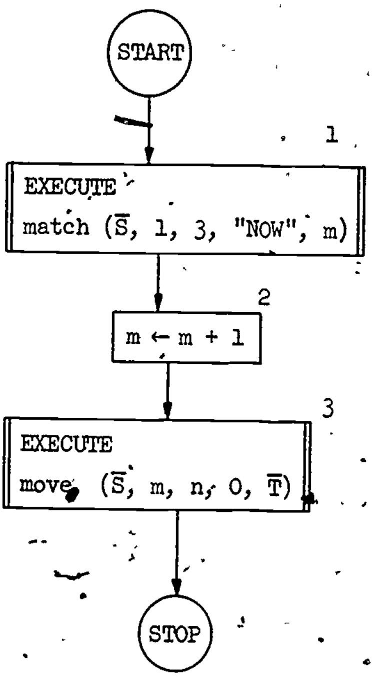
No answers for Exercises 7 - 10 inclusive are included in this edition.

7.

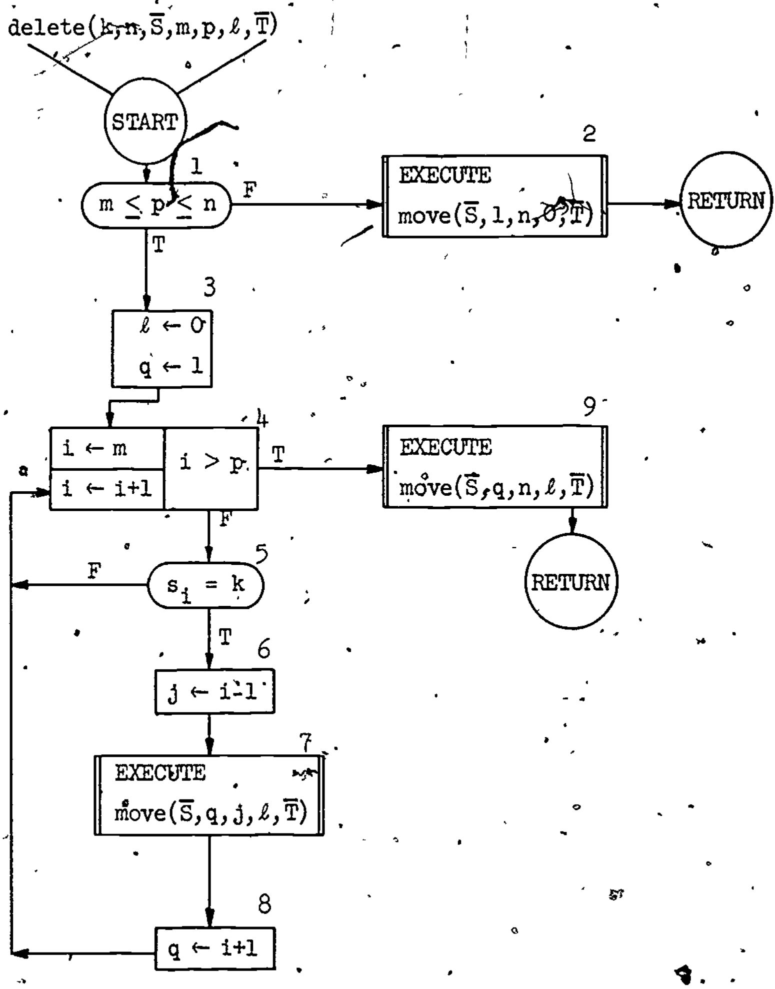

START
1
EXECUTE
chekch(n,$\overline{S}$,1,"Z",p)
2
p = 0
T
3
EXECUTE
move($\overline{S}$,1,n,0,$\overline{T}$)
STOP
F
4
q ← p-1
5
EXECUTE
move($\overline{S}$,1,q,0,$\overline{T}$)
6
EXECUTE
chekch(n,$\overline{S}$,p,"A",m)
7
m = 0
T
8
EXECUTE
move($\overline{S}$,p,n,ℓ,$\overline{T}$)
STOP
F
9
EXECUTE
delete(",",n,$\overline{S}$,p,m,ℓ,$\overline{T}$)
10
m ← m+1
11
EXECUTE
move($\overline{S}$,m,n,ℓ,$\overline{T}$)
STOP

In this exercise the length of the string will be changed if any deletion takes place. Therefore, we may as well decide to produce a new string $\overline{T}$ of length ℓ. The flow chart then uses procedures chekch, move and delete.

8.


START
1
EXECUTE
match ($\overline{S}$, 1, 3, "NOW", m)
2
m ← m + 1
3
EXECUTE
move ($\overline{S}$, m, n, 0, $\overline{T}$)
STOP

9.



An advantage of this flow chart is that the new string is $\overline{T}$ regardless of whether or not deletion has taken place or even if the index p is out of bounds. On the other hand, since transfer from $\overline{S}$ to $\overline{T}$ comes in bunches after a character to be deleted has been found, move must be called twice (Boxes 7 and 9). This is a case in which use of a procedure seems to have complicated the flow chart.

8-3 A language to be translated

This description is only as detailed as will be needed for the remaining discussion of this chapter. It is not a complete definition of a source language; but, as far as it goes, yields a language quite similar to many in actual use. Prohibition of such items as subscripts or signed variables in assignment statements is not due to any great problem in their processing, rather their presence would only further complicate the exposition of an already complex problem without adding great insight.

Classroom discussion of the "is there more data?" question should be held even though it is not explicitly treated in the text. The difficulty is that different computers will behave differently if they run out of input cards; some will just stop, others will automatically transfer control to some predetermined instruction. The only uniform way to treat the question is to require that a card follow the last data card and that that card contain some special coding (e.g., an asterisk in Column 1) which is testable by the program and indicates that there is no more data. Obviously, no data card can be allowed to have the same coding as the "end of data" card.

8-4 Prescan (the steps of a compiler).

In the discussion of Figure 8-10, the phrase "literal characters" is used. This simply means characters appearing in a source program which stand for themselves. We have encountered something similar in output strings earlier and we use the same notation here. Namely that:

"E" is the character E itself.

"END" is the string of characters END .

The text points out that regardless of whether or not a colon is found by the check procedure in Box 3 of Figure 8-13 incrementing q by one in Box 10 yields its desired value. The student is not asked why this is so. We can answer this "why" by recalling the object of Box 8 which is to find the first position of the string in which the letter "E" of the word "END" could occur. If the statement has a label, a colon will be found by Box 8 and q will point at that colon so that $q \leftarrow q + 1$ sets the pointer at the proper place to test for "E". If the statement does not have a label, Box 8 sets q to zero so that again incrementation makes q point to the first character of the statement where "E" might occur.

Sections 8-5 and 8-6 will assume some exposure to Example 1 of this section. It should be discussed for this reason if for no other. The purpose of the process proposed here is to replace whatever identifier the programmer may originally have written (whether long or short) with identifiers of uniform structure, e.g., lengths, so that each identifier can be thought of as occupying just one element of a string. In some languages the form of the internal identifiers can be used to encode information about them (e.g., whether they represent real or integer numbers). In the language of Chapter 8, all variables represent real numbers so that the need for this type of distinction does not arise.

Each of the examples in this section is presented in two stages--the general, or overall description followed by an implementation via detailed flow chart. The latter build on the material in Section 8-2. We carefully urge the student to skip the detail during the first reading so that you and he can get on to see the more interesting developments in Section 8-5. Time permitting, all can return to Section 8-4 for a more detailed look.

8-5 The Decomposition of Assignment Statements

The decomposition of assignment statements is usually thought of as the heart of the compilation process. Section 8-5 develops the background enabling us to state a rule for decomposition of assignments. In Section 8-6 we transform this rule into flow charts.

It should be remembered (again and again) that the assignment statement is assumed to have been processed through the prescan of Section 8-4. In particular we assume that only properly written statements are to be dealt with. No protection is provided in case the assignment statement is improperly formed although most actual compilers do provide such protection.

The key that makes decomposition practical is the use of a parenthesis--free way of writing an expression. A binary expression is normally written with the operator separating its two operands, that is:

$$a + b, \quad a - b, \quad a \times b, \quad a / b, \quad \text{or} \quad a \uparrow b .$$

This is called infix form. The text makes use of a different way of writing such expressions, called postfix form:

$$ab+, \quad ab-, \quad ab\times, \quad ab/, \quad \text{or} \quad ab\uparrow .$$

A third form also exists, called prefix form:

$$+ab, \quad -ab, \quad \times ab, \quad /ab, \quad \text{or} \quad \uparrow ab .$$

To illustrate prefix form:

$A + B \times C$ is written as $+ A \times B C$

$A \times B + C$ is written as $+ \times A B C$

$(A + B) \times C$ is written as $\times + A B C$

$A \times (B + C)$ is written as $\times A + B C$.

In either prefix or postfix form, parentheses are unnecessary to indicate the order. In prefix form, operations are performed in the order in which operators appear when scanning the expression from right to left. In postfix form, operations are performed in the order in which the operators appear when scanning the expression from left to right. In the text we have chosen to use the postfix form simply because people are accustomed to reading from left to right. Either scheme can be used in writing a computer program. The student is asked to complete a list of instructions for evaluation of the postfix form:

$$Z\ A\ B \times C + C\ 2 \uparrow /\ B\ A - -\ A\ 2 \uparrow B\ 2 \uparrow +\ 2 \uparrow / \leftarrow$$

The completed list is:

1. multiply A by B
2. to this add C
3. save that result (call it α) and square C
4. divide α by the square of C
5. save that result (call it β) and subtract A from B
6. subtract from β the difference B - A
7. save that result (call it γ) and square A
8. save that result (call it δ) and square B
9. add the δ and the square of B
10. square the result of Step 9
11. divide γ by the result of Step 10
12. assign the result of Step 11 to Z.

To find an automatic transformation to postfix form, a precedence table is presented in Figure 8-20. The upper half of the table contains arithmetic operators in an order that agrees with experience (and the presentation of Chapter 2) as to order of precedence. The lower half of the table contains symbols that are not generally thought of as operators. We call them "isolators" at first but we will discover it useful to assign them precedence values and for present purposes to call them operators. A simple example is then presented to motivate consideration of adjacent pairs of operators as the basis of a transformation rule.

A tentative (and incomplete) rule is then presented and tested against an example. During the test, displayed in Figure 8-21, we find Greek letters being used to denote intermediate results. The choice of Greek letters is purposeful to emphasize what will be discovered promptly: that it is important that an intermediate result be obviously identifiable as such and as something different from the other symbols in the string. Following the tentative rule we suddenly find that the Greek letters representing intermediate results appear in the generated string where the operations they represent already appear. Clearly something is wrong.

Additional examples are helpful.

Expression	Triple Substitution	Generated String
$(A \uparrow 2 + B \uparrow 2 + C \uparrow 2) \uparrow 2$	α for $A \uparrow 2$	$A2\uparrow$
$(\alpha + B \uparrow 2 + C \uparrow 2) \uparrow 2$	β for $B \uparrow 2$	$A2\uparrow\ B2\uparrow$
$(\alpha + \beta + C \uparrow 2) \uparrow 2$	α for $\alpha + \beta$	$A2\uparrow\ B2\uparrow\ +$
$(\alpha + C \uparrow 2) \uparrow 2$	β for $C \uparrow 2$	$A2\uparrow\ B2\uparrow\ +\ C2\uparrow$
$(\alpha + \beta) \uparrow 2$	α for $\alpha + \beta$	$A2\uparrow\ B2\uparrow\ +\ C2\uparrow\ +$
$\alpha \uparrow 2$	α for $\alpha \uparrow 2$	$A2\uparrow\ B2\uparrow\ +\ C2\uparrow\ +\ 2\uparrow$

One might think that an indefinite number of intermediate results might be produced.

This could happen for particular forms of expression, such as:

$$A \uparrow 2 + B \uparrow 2 \times (C \uparrow 2 + D \uparrow 2 \times (E \uparrow 2 + \ldots\ ))))).$$

However, in most situations, two or three temporary internal identifiers are found to be sufficient.

Another example illustrates the importance of using parentheses carefully if you want to indicate an ordering of operations.

Expression	Triple Substitution	Generated String
$A \times B + C \uparrow D$	α for $C \uparrow D$	$CD\uparrow$
$A \times B + \alpha$	β for $A \times B$	$CD\uparrow\ AB\times$
$\beta + \alpha$	α for $\beta + \alpha$	$CD\uparrow\ AB\times +$

Note that the commutativity of addition has been assumed. If, in practice, the sequence of operations is important, one should write $(A \times B) + C \uparrow D$ to insure that the addition is done in the desired order.

The student is asked to experiment and decide what to do when an operator pair has equal precedence; whether or not to select that operator pair. A rule could be developed for either choice. Externally the difference is in whether a series of operations of equal precedence (for example, $A + B + C + D$) is to be done left to right or right to left. Internally, choosing to select an operator pair under the condition of equal precedence (i.e., left to right execution) aids in the economization of temporary internal identifiers of intermediate results. This is demonstrated in the first of the additional examples given here.

Other decomposition algorithms

The algorithm which we develop here is by no means the only one used in the computing profession. Indeed it can hardly be construed as the best. We were motivated to present this one primarily because it is not necessary to digress for the purpose of introducing the concept of a "stack", which is a programming technique used in explaining the way most compilers function. A stack is a last-in, first-out storage device. Another name for a stack is "push-down store". You will find many references in the literature to "stack compilers". A readable account of one of these compilers for instance, can be found in the paper by Arthur B. Evans, Jr., "An Algol 60 Compiler", Annual Reviews of Automatic Programming, Pergamon Press, 1964.

Answers to Exercises 8-5 Set A

List the order in which you would do the operations for each of the following. Identify the reasons for your choice of orderings.

(a) $A + B \times C$

Multiply B by C then add A because multiplication has a higher precedence than addition.

(b) $A \times B + C$

multiply A by B then add C because multiplication has a higher precedence than addition.

(c) $(A + B) \times C$

add A and B then multiply by C because a parenthesized subexpression must be evaluated before multiplying.

(d) $A \times (B + C)$

add B and C then multiply by A because a parenthesized subexpression must be evaluated before multiplying.

(e) $A + B \uparrow C \times D$

first form B^C, multiply that by D, then add A because the order of precedence is exponentiation, multiplication, then addition.

(f) $A + B \uparrow (C \times D)$

multiply C by D, form $B^{C \times D}$, then add A because a parenthesized subexpression must be evaluated first, then the precedence of operations is exponentiation before addition.

(g - j) Answers are similar to those above.

Answers to Exercises 8-5 Set B

How would the expressions of Exercise 16(e) through (j) be written in postfix notation?

(e) $A + B \uparrow C \times D$ becomes $ABC \uparrow D \times +$

(f) $A + B \uparrow (C \times D)$ becomes $ABCD \times \uparrow +$

(g) $(A + B) \uparrow C \times D$ becomes $AB + C \uparrow D \times$

(h) $A \uparrow B + C \times D$ becomes $AB \uparrow CD \times +$

(i) $A \uparrow (B + C) \times D$ becomes $ABC + \uparrow D \times$

(j) $A \uparrow (B + C \times D)$ becomes $ABCD \times + \uparrow$

Answers to Exercises 8-5 Set C

1.	Statement	Triple Substitution	Generated String
(a)	$Z \leftarrow A + B \times C;$	α for $B \times C$	$BC \times$
	$Z \leftarrow A + \alpha;$	β for $A + \alpha$	$ABC \times +$
	$Z \leftarrow \beta$	$Z \leftarrow \beta$	$ZABC \times + \leftarrow$
(b)	$Z \leftarrow A \times B + C;$	α for $A \times B$	$AB \times$
	$Z \leftarrow \alpha + C;$	β for $\alpha + C$	$AB \times C +$
	$Z \leftarrow \beta;$	$Z \leftarrow \beta$	$ZAB \times C + \leftarrow$
(c)	$Z \leftarrow (A + B) \times C;$	α for $A + B$	$AB +$
	$Z \leftarrow \alpha \times C;$	β for $\alpha \times C$	$AB + C \times$
	$Z \leftarrow \beta$	$Z \leftarrow \beta$	$ZAB + C \times \leftarrow$
(d)	$Z \leftarrow A \times (B + C);$	α for $B + C$	$BC +$
	$Z \leftarrow A \times \alpha;$	β for $A \times \alpha$	$ABC + \times$
	$Z \leftarrow \beta;$	$Z \leftarrow \beta$	$ZABC + \times \leftarrow$
(e)	$Z \leftarrow A + B \uparrow C \times D;$	α for $B \uparrow C$	$BC \uparrow$
	$Z \leftarrow A + \alpha \times D;$	β for $\alpha \times D$	$BC \uparrow D \times$
	$Z \leftarrow A + \beta;$	γ for $A + \beta$	$ABC \uparrow D \times +$
	$Z \leftarrow \gamma;$	$Z \leftarrow \gamma$	$ZABC \uparrow D \times + \leftarrow$
(f)	$Z \leftarrow A + B \uparrow (C \times D);$	α for $C \times D$	$CD \times$
	$Z \leftarrow A + B \uparrow \alpha;$	β for $B \uparrow \alpha$	$BCD \times \uparrow$
	$Z \leftarrow A + \beta;$	γ for $A + \beta$	$ABCD \times \uparrow +$
	$Z \leftarrow \gamma;$	$Z \leftarrow \gamma$	$ZABCD \times \uparrow + \leftarrow$
(g)	$Z \leftarrow (A + B) \uparrow C \times D;$	α for $A + B$	$AB +$
	$Z \leftarrow \alpha \uparrow C \times D;$	β for $\alpha \uparrow C$	$AB + C \uparrow$
	$Z \leftarrow \beta \times D;$	γ for $\beta \times D$	$AB + C \uparrow D \times$
	$Z \leftarrow \gamma;$	$Z \leftarrow \gamma$	$ZAB + C \uparrow D \times \leftarrow$
(h)	$Z \leftarrow A \uparrow B + C \times D;$	α for $A \uparrow B$	$AB \uparrow$
	$Z \leftarrow \alpha + C \times D;$	β for $C \times D$	$AB \uparrow CD \times$
	$Z \leftarrow \alpha + \beta;$	γ for $\alpha + \beta$	$AB \uparrow CD \times +$
	$Z \leftarrow \gamma;$	$Z \leftarrow \gamma$	$ZAB \uparrow CD \times + \leftarrow$

(i)	$Z\leftarrow A\uparrow(B+C)\times D;$	α for	$B+C$	$BC+$
	$Z\leftarrow A\uparrow\alpha\times D;$	β for	$A\uparrow\alpha$	$ABC+\uparrow$
	$Z\leftarrow\beta\times D;$	γ for	$\beta\times D$	$ABC+\uparrow D\times$
	$Z\leftarrow\gamma;$		$Z\leftarrow\gamma$	$ZABC+\uparrow D\times\leftarrow$
(j)	$Z\leftarrow A\uparrow(B+C\times D);$	α for	$C\times D$	$CD\times$
	$Z\leftarrow A\uparrow(B+\alpha);$	β for	$B+\alpha$	$BCD\times+$
	$Z\leftarrow A\uparrow\beta;$	γ for	$A\uparrow\beta$	$ABCD\times+\uparrow$
	$Z\leftarrow\gamma;$		$Z\leftarrow\gamma$	$ZABCD\times+\uparrow\leftarrow$

2.

Statement	Triple Substitution	Generated String
$Z\leftarrow((A\times B+C)/C\uparrow2-(B-A))$ $/(A\uparrow2+B\uparrow2)\uparrow2;$	α for $A\times B$	$AB\times$
$Z\leftarrow((\alpha+C)/C\uparrow2-(B-A))$ $/(A\uparrow2+B\uparrow2)\uparrow2;$	β for $\alpha+C$	$AB\times C+$
$Z\leftarrow(\beta/C\uparrow2-(B-A))$ $/(A\uparrow2+B\uparrow2)\uparrow2;$	γ for $C\uparrow2$	$AB\times C+C2\uparrow$
$Z\leftarrow(\beta/\gamma-(B-A))$ $/(A\uparrow2+B\uparrow2)\uparrow2;$	δ for β/γ	$AB\times C+C2\uparrow/$
$Z\leftarrow(\delta-(B-A))/(A\uparrow2+B\uparrow2)\uparrow2;$	ϵ for $B-A$	$AB\times C+C2\uparrow/BA-$
$Z\leftarrow(\delta-\epsilon)/A\uparrow2+B\uparrow2)\uparrow2;$	ϕ for $\delta-\epsilon$	$AB\times C+C2\uparrow/BA--$
$Z\leftarrow\phi/(A\uparrow2+B\uparrow2)\uparrow2;$	ρ for $A\uparrow2$	$AB\times C+C2\uparrow/BA--A2\uparrow$
$Z\leftarrow\phi/(\rho+B\uparrow2)\uparrow2;$	μ for $B\uparrow2$	$AB\times C+C2\uparrow/BA--A2\uparrow B2\uparrow$
$Z\leftarrow\phi/(\rho+\mu)\uparrow2;$	ν for $\rho+\mu$	$AB\times C+C2\uparrow/BA--A2\uparrow B2\uparrow+$
$Z\leftarrow\phi/\nu\uparrow2;$	σ for $\nu\uparrow2$	$AB\times C+C2\uparrow/BA--A2\uparrow B2\uparrow+2\uparrow$
$Z\leftarrow\phi/\sigma;$	λ for ϕ/σ	$AB\times C+C2\uparrow/BA--A2\uparrow B2\uparrow+2\uparrow/$
$Z\leftarrow\lambda;$	$Z\leftarrow\lambda$	$ZAB\times C+C2\uparrow/BA--A2\ B2\uparrow+2\uparrow/\leftarrow$

Yes!

SUPPLEMENT

The Tree Search with Applications to the Four Color Problem and Games

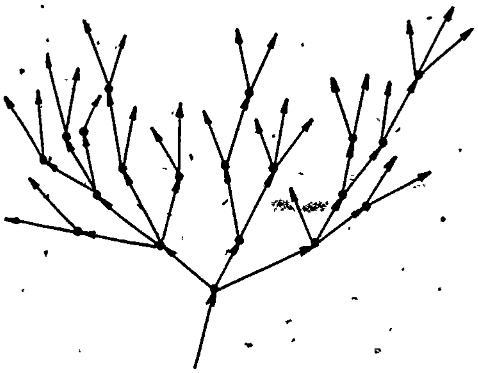
This unit is provided for the teacher who may have gifted or enthusiastic students who want to try their mettle on some really hard algorithms. Also, this material could be used as the basis for supplementary lectures. The unit does not use functions or procedures and hence could be inserted any time after the completion of Chapter 4.

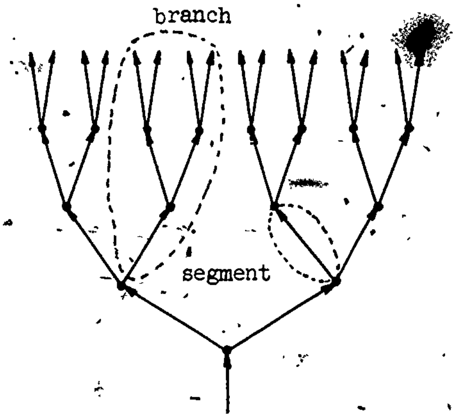
Since the exercises in Chapter 6 and the first five sections of Chapter 7 are rather few in number, this unit could be used as a project to fill that dead spot.

If the students are to be asked to develop the four coloring algorithm, it is suggested that the material up through the reduced connection table be duplicated and given them. In the case of the game of "31", the student should be given the material up through the rules of the game.

The Tree Search

Two examples of trees are shown below.



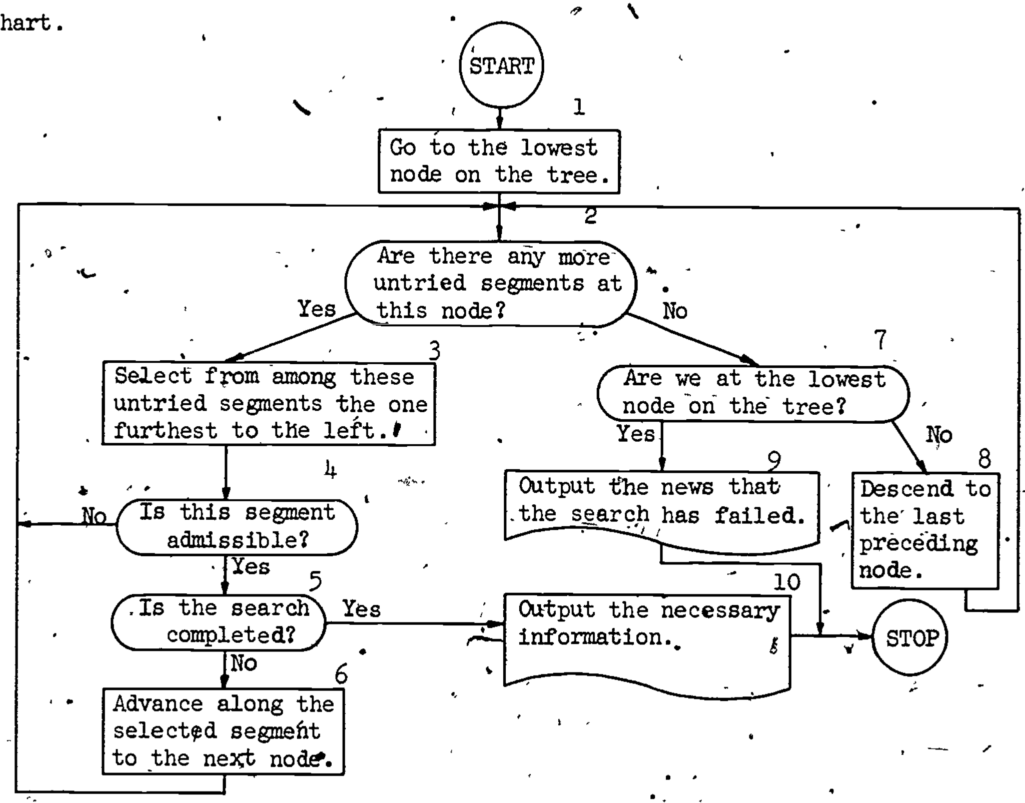


We see that there are a number of "nodes" on these trees with one branch coming into each node and two or more branches emanating. In further pictures we will omit the arrows and assume it to be understood that the direction of growth is upward.

By a segment we will mean one of the vectors reaching from one node to the next one. By a branch we mean a segment together with everything that follows it. Thus, a branch is a "sub-tree" of the original tree. In climbing a tree, when we arrive at a given node we choose one of the segments emanating from it, advance along this segment to the next node and again choose a segment, repeating this process until we reach the top. We see that we can reach the top at many different points. However, once we have reached a certain point at the top, there is only one path by which we may descend.

A number of mathematical problems and a great many games have the tree search as a model. In a tree search we attempt to find a path up the tree to one of a number of segments on which a desirable object is located. Since, in most applications, the number of paths is formidably large, the task seems quite discouraging. However, it turns out that in the applications there are certain segments which we are barred from choosing. (The inadmissibility of a segment can only be determined after we have climbed to that position.) Since a forbidden segment removes a whole branch from consideration, the size of the problem may be drastically reduced.

We next present a systematic procedure, i.e., an algorithm, for conducting a tree search. Our actions are most easily described in the following flow chart.


START
1
Go to the lowest node on the tree.
2
Are there any more untried segments at this node?
Yes
No
3
Select from among these untried segments the one furthest to the left.
4
No
Is this segment admissible?
Yes
5
Is the search completed?
Yes
No
6
Advance along the selected segment to the next node.
7
Are we at the lowest node on the tree?
Yes
No
9
Output the news that the search has failed.
8
Descend to the last preceding node.
10
Output the necessary information.
STOP

Generalized flow chart for a tree search

In applications, some of the boxes in this flow chart (especially Box 4) may be broken down into a number of flow chart components.

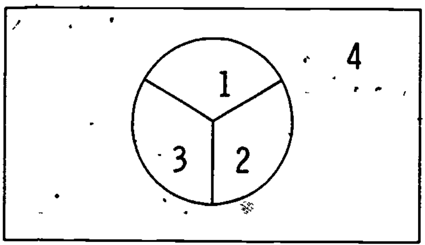
Our first application of a tree search is the problem of determining whether a given map can be colored with four colors. This problem is modeled by a tree with four segments emanating from each node. Each path up the tree represents a coloring of the entire map. The forbidden paths are those which would result in some country having the same color as a neighboring country. The desired object is to reach the very top of the tree along a permissible path.

Before developing this algorithm we provide a discussion of the four color problem. Much of this material is background material and may be skipped if desired.

<u>The Four Color Problem.</u>

Maps are colored so as to make it easy to see at a glance the extent of each country. It is clearly necessary that neighboring countries (i.e., countries with a common boundary line) should be assigned different colors. This is the only requirement we impose on the coloring of maps.

A checkerboard is an example of a map which can be colored with only two colors. The four country map shown below requires four colors. The reason for this is that, each pair of countries being adjacent, no two can have the same color.



It didn't take us long to find a map requiring four colors. Yet, in over a hundred years of searching, no one has succeeded in finding a map requiring five. It is natural to conjecture that every possible map can be colored with four colors and many mathematicians have racked their brains trying to prove this conjecture. The best they have been able to do so far is to show that every map can be colored using no more than five colors. [In fact, a very simple proof of the five color theorem exists which is quite suitable for presentation to high school students. See, for example, Courant and Robbins, What is Mathematics?]

We are about to see how computer methods can be applied to the four color problem. If the four color conjecture is true, a computer will not be able to devise a proof of it. If, on the other hand, the four color conjecture is false, a computer might be able to find that out. In particular, for a given map, a computer can determine whether it can be colored in four colors. That is the task for which we will construct an algorithm.

Before we present this algorithm, a few remarks concerning the coloring of maps may be helpful.

A minimal five color map is a map requiring five colors and such that every other map requiring five colors has at least as many countries. It is easy to show that if maps requiring five colors do indeed exist, then there exist minimal five color maps satisfying:

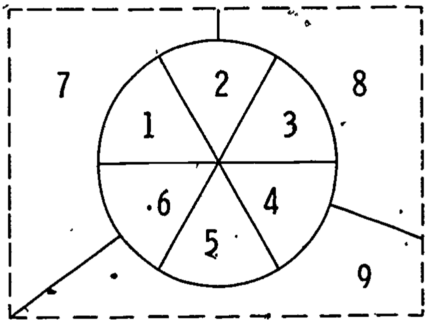
i) no point is a boundary point of more than three countries; and

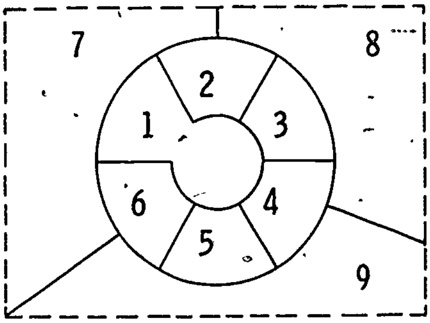
ii) each country is a neighbor of at least five others. [Every minimal map must satisfy this condition.]

It is customary to consider as candidates for counter-examples to the four color conjecture only maps fulfilling these conditions. We do not need to use these conditions in our algorithm but their proof is established by the following map fragments.

Digression: Establishing Condition (i) and (ii)

To establish the legitimacy of requiring condition (i), suppose that the map fragment on the left is a part of a minimal five color map.

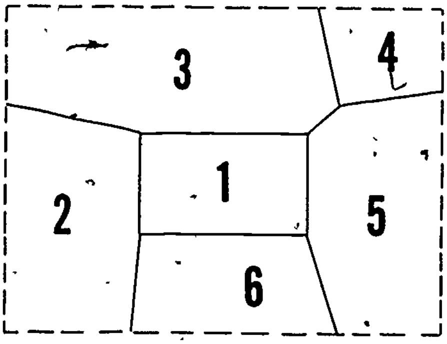


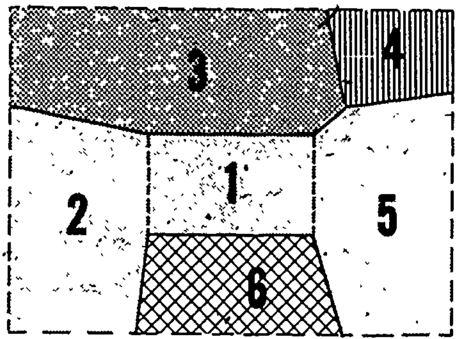


Condition (i) is clearly violated since the hub of the wheel is a boundary point of six countries.

But that objection is eliminated by the modification shown on the right. This map has the same number of countries as the original. All countries which were neighbors before the modification are still neighbors, but now country 1 has three new neighbors. Clearly, this modified map will require no fewer colors than the original. Thus, if the original map was a minimal five color map, so is the modified map. This same modification can be made at all points which are boundary points of more than three countries, thus establishing property (i).

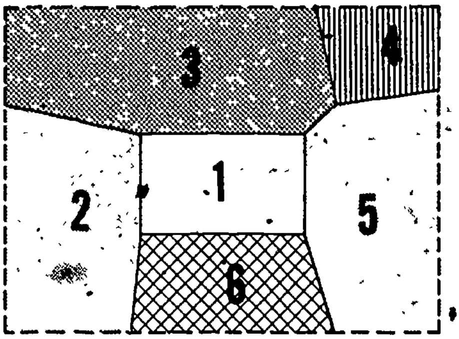
To establish the legitimacy of requiring property (ii), suppose that the fragment on the left below is a fragment of a larger map which is a minimal five color map.

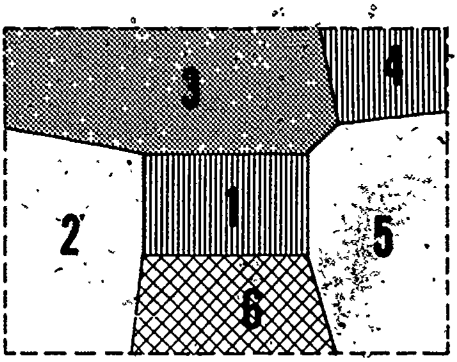




Here condition (ii) is violated as country 1 has just four neighbors. Now we obliterate the boundaries between 1 and 2 and between 1 and 5, thus making countries 1, 2 and 5 into a single country in the map on the right. This map has two fewer countries than the original. Since the original was a <u>minimal</u> five color map, the revised map must be four colorable. We suppose this to have been done in the figure on the right above.

Next we restore the deleted boundaries and uncolor country 1 as shown in the figure on the left below. In this map country 1 has four neighbors but these neighbors have only three <u>different</u> colors, since 2 and 5 are colored the same.

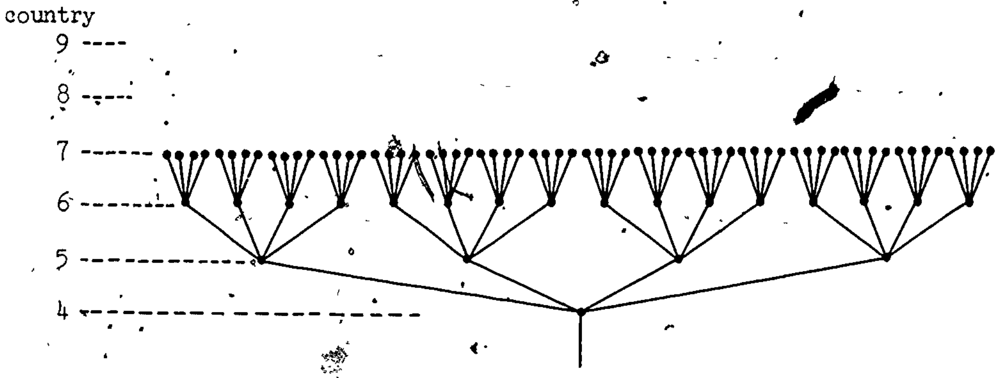




Thus, the fourth color is available for country 1 as seen in the figure on the right. And now the whole map is four colored. This contradicts our assumption that the original map was a minimal five color map and hence establishes property (ii). [If countries 2 and 5 have a common boundary outside this fragment, then in the entire discussion use 3 and 6 instead of 2 and 5.]

Four Coloring as a Tree Search

We will model the problem of finding a four coloring of a given map as a tree search. The tree will have four segments emanating from each node. Each country (except for the first three) will have a corresponding level of nodes.



The segments emanating from the nodes at the n th level represent the four possible colorings of the n th country. The particular node which we have reached at the n th level represents the history of the colors presently assigned to lower numbered countries.

The inadmissible segments are those which would result in a country being given the same color as a previously colored (lower numbered) neighboring country. Which countries are neighbors depends on the particular map and will have to be input as data.

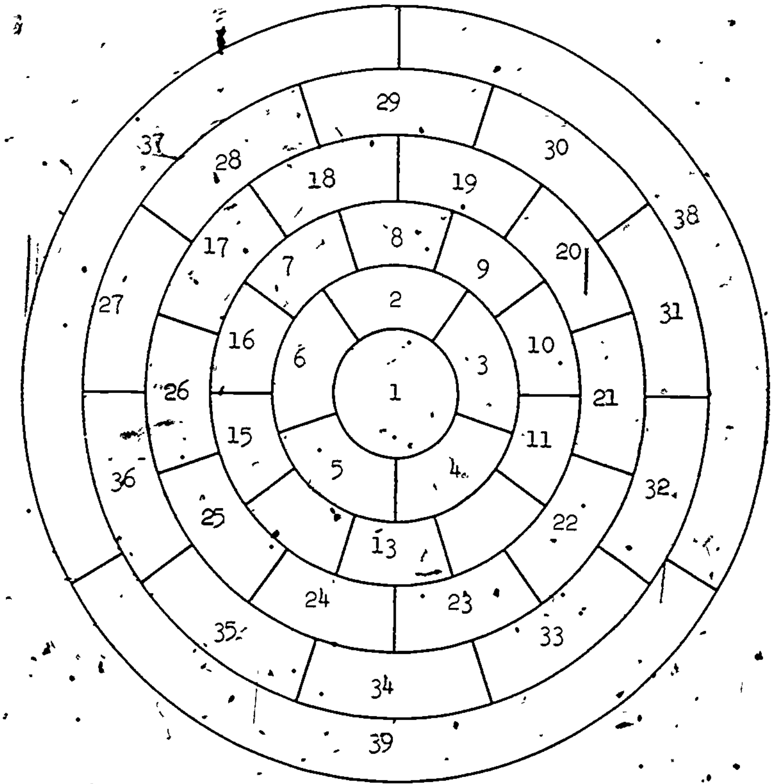
The object of the search is to find a path from the bottom of the tree to the top in which every segment is admissible.

Preparation for the Algorithm

It will simplify matters greatly to follow the ensuing discussion using the map provided in the next figure as an example. The first step in preparing for the algorithm is the numbering or indexing of the countries. As you see, this has already been done.

The efficiency of the algorithm will be greatly improved if each country borders on that with the next lower number and on at least one other country with a lower number. We do not absolutely insist on this.

We do, however, require that the first three countries all be neighbors of each other.



Example of map to be four colored

When the numbering has been done we construct the "connection table", listing after each country all of its neighbors in increasing order.* This is shown for our example in Table I. Next we construct the "reduced connection table" by striking out of each row in the table all numbers greater than the number of the row. The reduced connection table for our example is seen in Table II. The subscripted variable w_i gives the width of the i^{th} row. We now have all the input required for our algorithm.

Table I. The Connection Table for the Example

Region	Neighbors						Region	Neighbors						
1.	2	3	4	5	6		21.	10	11	20	22	31	32	
2.	1	3	6	7	8	9	22.	11	12	21	23	32	33	
3.	1	2	4	9	10	11	23.	12	13	22	24	33	34	
4.	1	3	5	11	12	13	24.	13	14	23	25	34	35	
5.	1	4	6	13	14	15	25.	14	15	24	26	35	36	
6.	1	2	5	7	15	16	26.	15	16	17	25	27	36	
7.	2	6	8	16	17	18	27.	17	26	28	36	37		
8.	2	7	9	18	19		28.	17	18	27	29	37		
9.	2	3	8	10	19	20	29.	18	19	28	30	37	38	
10.	3	9	11	20	21		30.	19	20	29	31	38		
11.	3	4	10	12	21	22	31.	20	21	30	32	38		
12.	4	11	13	22	23		32.	21	22	31	33	38	39	
13.	4	5	12	14	23	24	33.	22	23	32	34	39		
14.	5	13	15	24	25		34.	23	24	33	35	39		
15.	5	6	14	16	25	26	35.	24	25	34	36	39		
16.	6	7	15	17	26		36.	25	26	27	35	37	39	
17.	7	16	18	26	27	28	37.	27	28	29	36	38	39	
18.	7	8	17	19	28	29	38.	29	30	31	32	37	39	
19.	8	9	18	20	29	30	39.	32	33	34	35	36	37	38
20.	9	10	19	21	30	31								

*Actually, the connection table is not used. Its only purpose is in clarifying the presentation of the "reduced connection table". The reduced connection table could be presented directly.

Table II. Reduced Connection Table for the Example

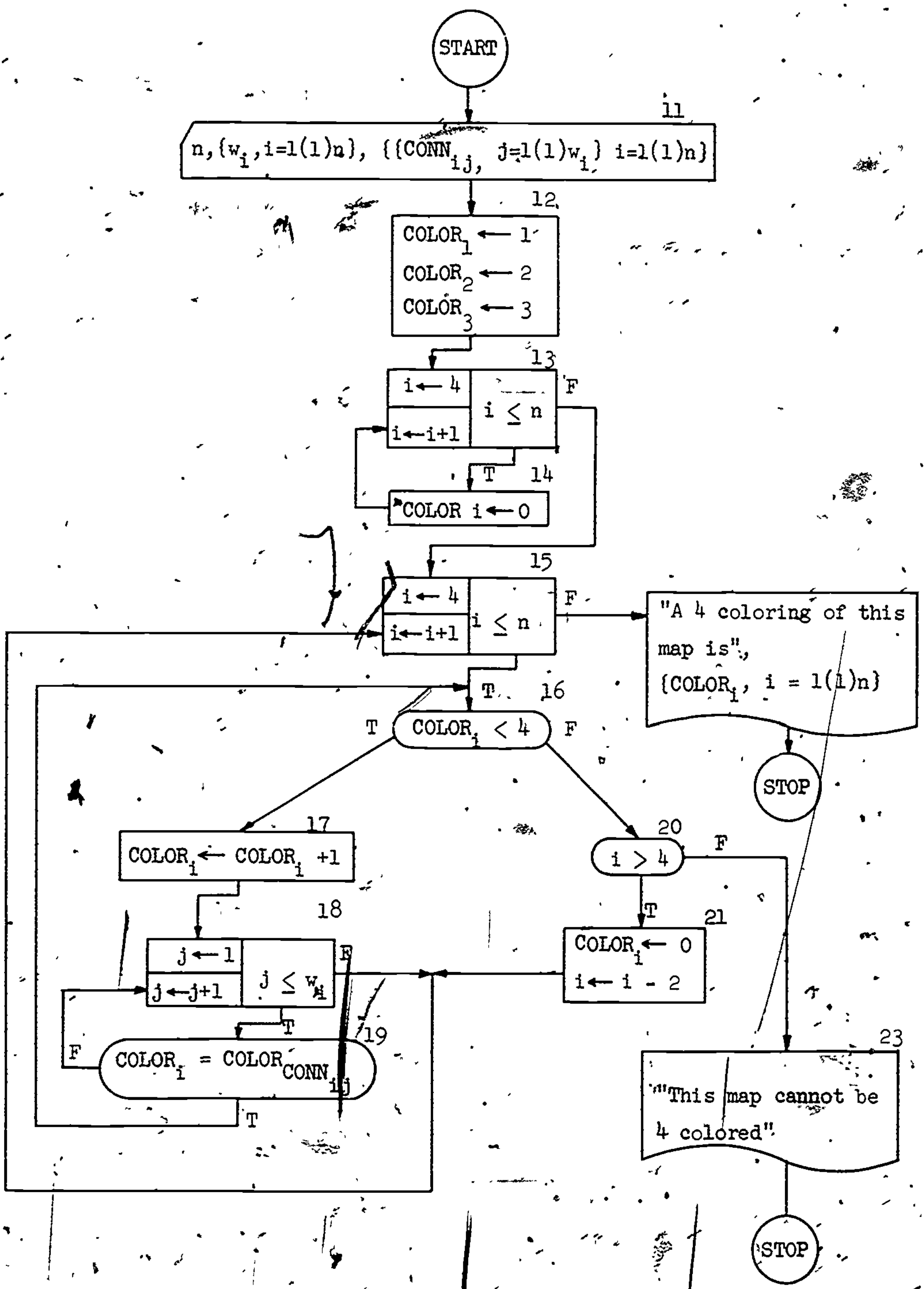
country i	neighbors $CONN_{ij}$	width w_i	country i	neighbors $CONN_{ij}$	width w_i
1		0	21	10 11 20	3
2	1	1	22	11 12 21	3
3	1 2	2	23	12 13 22	3
4	1 3	2	24	13 14 23	3
5	1 4	2	25	14 15 24	3
6	1 2 5	3	26	15 16 17 25	4
7	2 6	2	27	17 26	2
8	2 7	2	28	17 18 27	3
9	2 3 8	3	29	18 19 28	3
10	3 9	2	30	19 20 29	3
11	3 5 10	3	31	20 21 30	3
12	4 11	2	32	21 22 31	3
13	4 5 12	3	33	22 23 32	3
14	5 13	2	34	23 24 33	3
15	5 6 14	3	35	24 25 34	3
16	6 7 15	3	36	25 26 27 35	4
17	7 16	2	37	27 28 29 36	4
18	7 8 17	3	38	29 30 31 32 37	5
19	8 9 18	3	39	32 33 34 35 36 37 38	7
20	9 10 19	3			

The Four Coloring Algorithm

We are now prepared to present the four coloring algorithm seen in the accompanying flow chart. The boxes are numbered starting with 11 for ease of comparison with the generalized flow chart for a tree search.

Box 11 is our input. It gives the number of countries, the reduced connection table and its widths.

$COLOR_i$ represents the color (1, 2, 3 or 4) tentatively given to the i^{th} country. Since countries 1, 2, and 3 are mutual neighbors, they must have distinct colors, so we arbitrarily assign $COLOR_1$, $COLOR_2$ and $COLOR_3$ the values 1, 2 and 3 in flow chart Box 12. In Boxes 13 and 14 the rest of the countries are blanked.



Four coloring a map

On emerging from the F exit of Box 13 the tree search begins. The rest of the flow chart is an exact parallel of the generalized tree search flow chart. In the ensuing discussion of the four coloring algorithm we will give parenthetical references to the corresponding boxes of the tree search flow chart.

In the initialization portion of Box 15 (Box 1) we initialize the value of the variable i at 4. This variable should be regarded as a pointer which tells us which country we are attempting to color (which level of nodes we are at).

In Box 16 (Box 2) we make a test to determine whether any more colors are available to be tried for the i^{th} country. On emerging from the T exit we tentatively assign the next numbered color to the i^{th} country in Box 17 (Box 3). Now we go through the loop in Boxes 18 and 19, (this loop corresponds to Box 4) to determine whether the color assigned in Box 17 is admissible. That is, we check to see whether this color has been given to any of the previously colored neighbors of the i^{th} country.

If this color has been used on a neighbor we will exit from the loop on the T of Box 19 back to the test in Box 16 where we check whether another color is available to try for the i^{th} country. If, on the other hand, we find that this color has not been used on a neighbor, we emerge from the loop at the F of Box 18, increment i by 1 in the incrementation portion of Box 15 (Box 6) and proceed to the coloring of the next country.

If at any time we find in Box 16 (Box 2) that no more colors are available to be tried for the i^{th} country, then we have to go back to the previous country (go back one node down the tree). First we check in Box 20 (Box 7) whether we are already at the base of the tree. If we are at the base of the tree, we output the news that the search has failed in Box 23 (Box 9). If we are not at the base of the tree, we uncolor the i^{th} country in Box 21 and go back one node. This stepping back one node is accomplished by stepping back two in Box 21 and up one in the incrementation portion of Box 15. (These together correspond to Box 8.)

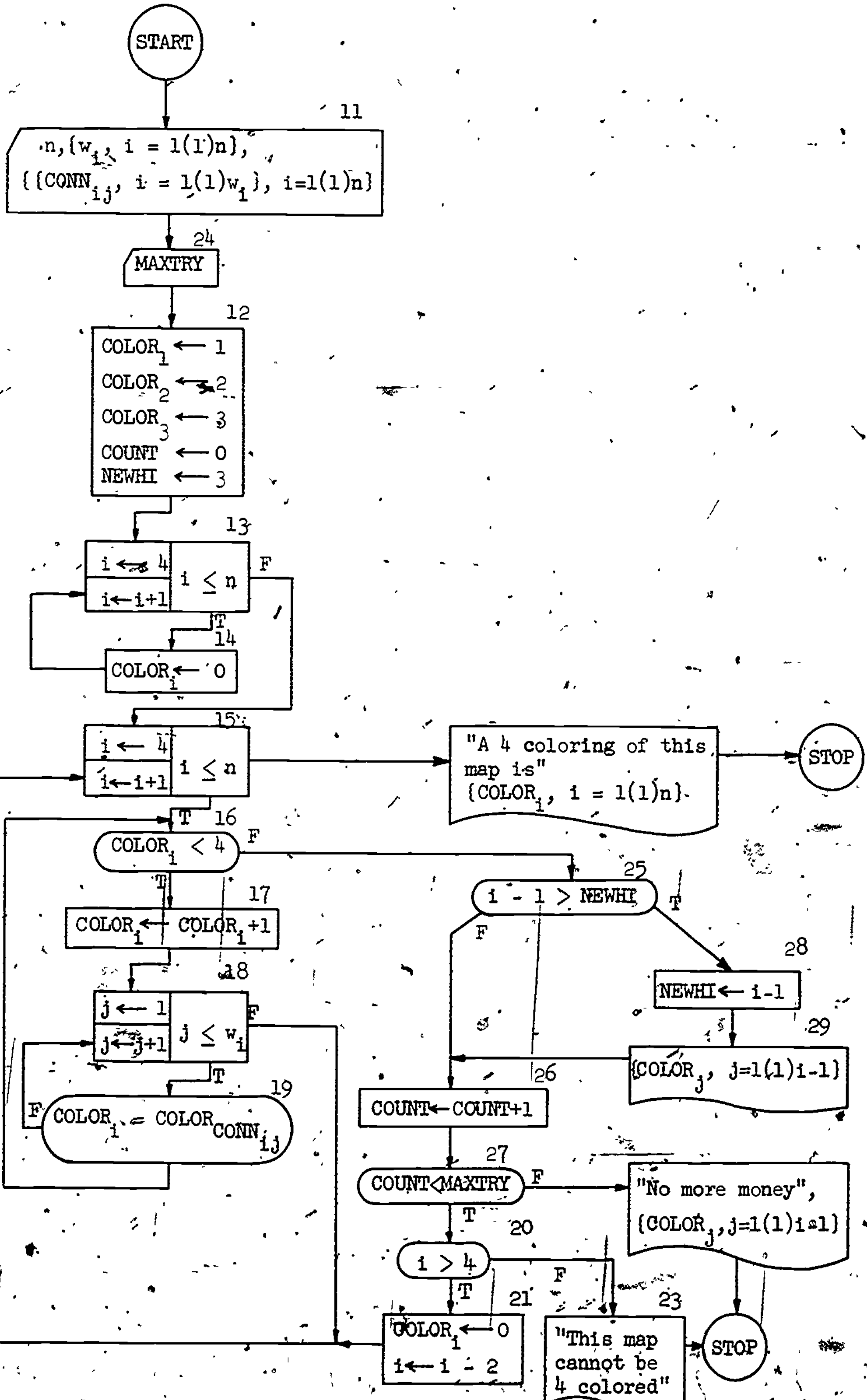
If we ever emerge from the F exit of Box 15 (the test portion of Box 15 corresponding to Box 5) then the vector, COLOR, contains an admissible coloring of the entire map which we print out in Box 22 (Box 10).

Improvements and Modifications

The greatest fault in the algorithm occurs when we have not succeeded in numbering the countries so that each borders on its predecessor. In this case, there is little use in stepping down the tree one branch at a time. We may as well drop down until we first reach a neighbor of the i^{th} country. For only these neighbors can contribute to the blocking of the coloring of the i^{th} country. Unfortunately, the incorporation of this feature immensely complicates the algorithm and we will not discuss such a modification here.

Another drawback of this algorithm is that we cannot predict in advance how long the program will require for execution on a computer. We might then decide to cut the process off after the millionth or MAXTRY^{th} setback in Box 16. But then we would have no information at all from our program. We might therefore decide to print out the present state of the coloring process before any setback in which the number of countries colored has reached a new high, NEWHI.

These features are incorporated in our final modification of the four color flow chart. The modifications are seen in Box 12 and Boxes 24-30.



Modified 4 coloring algorithm

Tree Games

A great many games are modeled by trees. We will call these games "tree games". Among these tree games are chess, checkers, go, nim and tic-tac-toe.

In tree games each segment represents a move of play by one player or the other. Each level of nodes corresponds to the number of elapsed moves. The inadmissible segments represent illegal moves. Each admissible path from the base up to a given node represents a partially completed game.

One computer problem associated with tree games is that of "teaching the computer to play the game". This problem is interesting as it relates to human thought processes. At this time considerable success has been achieved in teaching machines to play checkers and somewhat less in teaching them to play chess.

But that is not the problem we will consider here. We will consider the problem of determining the outcome of the game in the event of best play by both players. (There are other ways of analyzing this problem, which, for some games, may be considerably better.)

Tree games all have the property that if both players were omniscient and could see all the possible consequences of all the moves they could make (as is actually the case with tic-tac-toe), then the result of the game would be foreordained as a win for the first player, a win for the second player, or a draw.

For any of these games algorithms can be devised for determining the predestined result. However, most tree games are either too easy or too difficult for computer analysis. For the games of chess or go the problem is ridiculuously beyond the powers of a computer. The highest speed computer working full time for a billion years would not scratch the surface of the problem. On the other hand, the games of nim and tic-tac-toe are sufficiently simple that rules can be given for the strategy leading to the best result (a draw in tic-tac-toe and a win for the first or second player in nim depending only on the initial numbers of sticks in the piles).

The tree game we have chosen to analyze here is very simple but it will illustrate the main ideas. This game has the pedagogical advantage that the best strategy is not widely known. The name of the game is "31". The rules are quite trivial. A single die is rolled and the number that comes up is recorded. Then each of the two players proceeds in turn to turn up the number of his choice, except that the numbers currently on the top and bottom are inadmissible. The numbers that the players turn up are all added to a single running total which has as its initial value the number rolled at the beginning of the game. A player bringing this total to exactly 31 wins the game; a player exceeding 31 loses.

We see two ways in which the game differs from some other games. First, no draw is possible; second, a player can win or lose at his own turn. The number 31 is quite arbitrary and could be changed to any other number.

Tree games differ from other tree searches in that a result which is desirable for one player is undesirable for the other. This results in the characteristic pattern of analysis. First, a path is followed up the tree until a conclusion is reached. Then the losing player takes back his last move and tries another move. This game is played to its conclusion, the loser retracting his last move, etc. These move retractions will eventually carry us to the base of the tree. When a retraction would carry us below the base of the tree, the most recent winner has an incontrovertible win as his opponent has exhausted all of his resources for improvement.

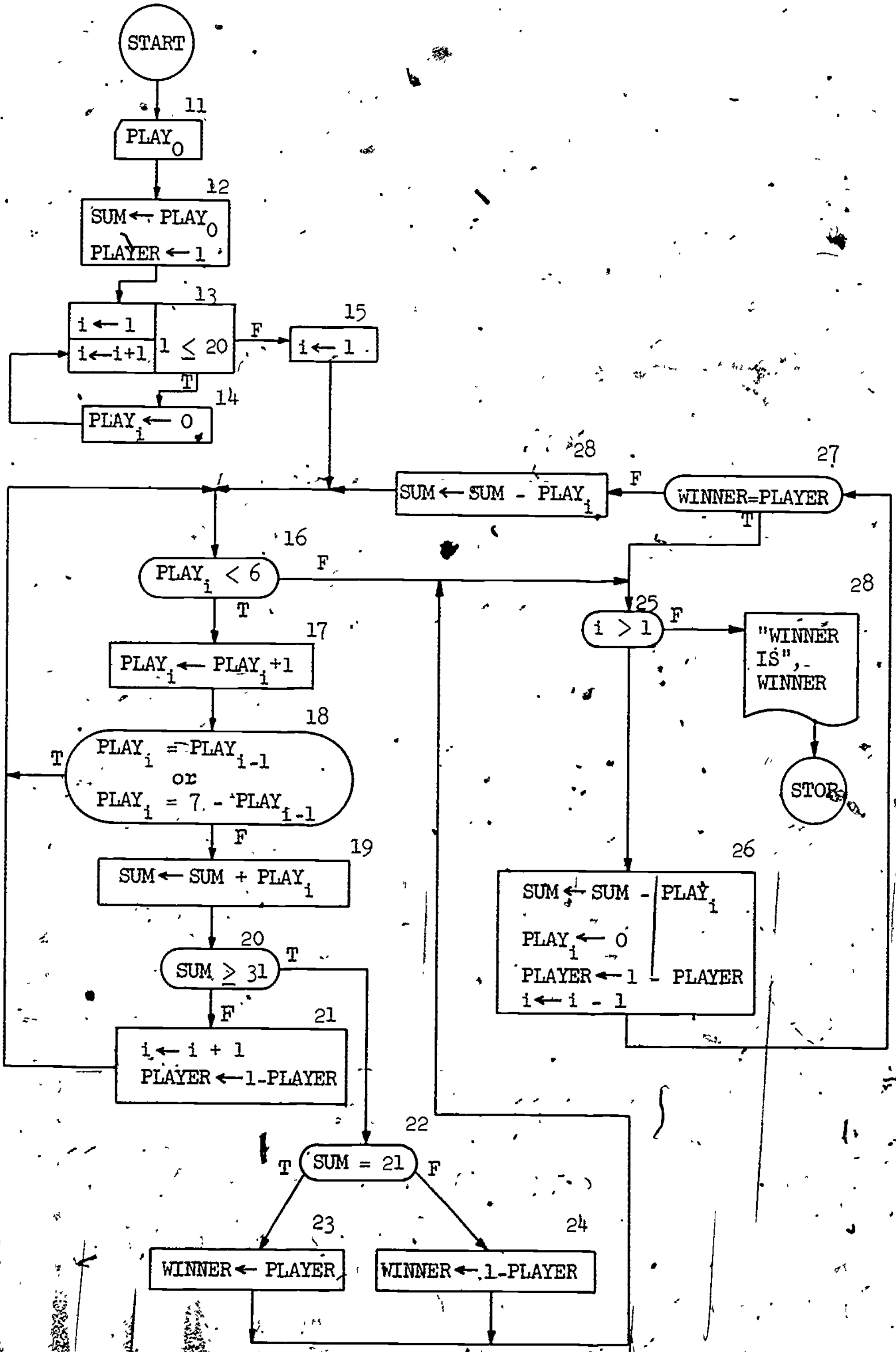
In the flow chart which follows, the two players are numbered 1 and 0, 1 playing first. It is easiest to consider that at each turn there are 6 possible plays, two of which are inadmissible.

Again the algorithm is analogous to the generalized tree search algorithm except that the output in Box 10 is replaced by climbing down the tree to let the loser have another try. In discussing the following flow chart we again make parenthetical references to the generalized tree search algorithm. An abbreviated description seems to be in order, in view of the similarities with the previous examples.

The variable i denotes, from Box 15 onward, the number of the move to be made (the level of nodes at which we are located). The vector PLAY records all the moves which have been made in arriving at the present position. The variable PLAYER assumes the value 1 for odd numbered moves and 0 for even numbered moves and indicates which player has the move. SUM and WINNER are self-explanatory.

In Boxes 11-14 the game is being set up for play. The initial roll of the die, $PLAY_0$, is input and the variables SUM, PLAYER and PLAY are initiated. In Box 15 (Box 1) the search commences. In Box 16 (Box 2) we test to see whether any more moves are available to be tried at the i^{th} turn. If so, we select the next numbered die face in Box 17 (Box 3) and test for legality in Box 18 (Box 4) remembering that the sum of the numbers on opposite faces of a die is always equal to 7. After augmenting the running total in Box 19 we test to see whether a game has been completed. (Boxes 19 and 20 are equivalent to Box 5.)

ST



Analyzing the game of "31"

If the game is not completed, the privilege of moving is given to the opponent in Box 21 (Box 6) and play continues. Boxes 22-24 are a substitute for Box 10. Here, if the game was seen to be completed in Box 20, we designate the appropriate player as winner. [It is unnecessary to allow the player who has just moved to retract his move and try again in the case that he has lost, for any moves he has left will also lose. This feature is peculiar to this particular game.]

In the event that a player has run out of moves in Box 16 or that a winner has been designated in Box 23 or 24, we must climb back down the tree. First we must determine in Box 25 (Box 7) whether there are any earlier moves to return to. If there are, we step back one move in Box 26 (Box 8) after first erasing the effect of the last made move which now lies on a different branch. The move also changes hands in this box.

It will be noted that we stepped down just one node instead of going directly to the last previous move of the loser. Box 27, which has no equivalent in the tree search flow chart, checks, before resuming play, whether we have in fact stepped back far enough. On exiting from Box 27 on F the option ~~of moving~~ reposes with the loser and we proceed ~~to Box 28.~~ It must be remembered that another move is to be substituted for the i^{th} move. The last made i^{th} move is therefore first removed from SUM in Box 28. PLAY$_i$ must not be erased here as the loser must have a record of the last move he tried at this point.

When the test is failed in Box 25, so that the loser has no earlier moves left to retract, the search is ended and the result is printed in Box 25 (Box 9).

The algorithm can, of course, be modified to provide for the output of additional information as desired.

Revealing the winning strategy of the game of "31" will ruin it as a computer problem but we will whisper this strategy to the teacher. You have a winning position if you can leave your opponent with the SUM ≡ 4 mod 9, or, if this is impossible, with the SUM ≡ 0 or 5 mod 9 with a 3 or 4 faced up. If you can leave your opponent with such a position, he will be unable to do the same to you, nor will he be able to prevent you from again achieving such a position on your next turn.

If the play recommended above will result in your exceeding 31, then you will be able instead to bring the total to 30 with a 1 faced up, which you do.

From the above it follows that if the initial roll is 4, then the game is won for the second player. With any other initial roll the game is won for the first player.

STUDY GUIDE IN DIGITAL COMPUTING AND RELATED MATHEMATICS*

1. THE PURPOSE OF THIS GUIDE

Introducing the subject of digital computers into the high schools involves knowledge, materials, and points of view that are at present not normally part of the training of the high school mathematics teacher. In order that the subject be introduced as widely and rapidly as possible, an in-service training program is necessary. Teachers must acquire an understanding of the new concepts to use them effectively and confidently in the classroom.

The purpose of this Study Guide is to aid the teacher in acquiring a familiarity with digital computer concepts or to further his knowledge of the field. The concept of an algorithm is stressed in the suggested materials, since it is basic to the mathematical solution of many problems. An algorithm ~~is a list of instructions~~ specifying a finite sequence of operations whose execution will yield the answer to a particular problem or class of problems. Algorithms may be stated in diagrammatic form or as computer programs. The programs are themselves sequences of operations for computer processing. Colleges are stressing the algorithmic approach to mathematics. Thus, it is important that the high school student study this concept, whether he writes computer programs or not.

The use of this Guide should not preclude the use of other sources of information. For example, the teacher would profit from a course in computer programming, preferably taught at a college. In such a course he would normally gain laboratory experience with a computer. While contact with the computer is not essential to an understanding of the use of the machine, it greatly enhances the training. In the event the teacher cannot conveniently enroll in a college course to strengthen his study program, he is urged to seek computer time at a nearby college or at a government or industrial research organization.

Professional and scientific organizations, specifically the Association for Computing Machinery, are excellent sources of additional information and advice on professional, educational, and vocational aspects of computing. Members of local chapters of this organization are usually very helpful in providing advice

*This study guide was published originally by SMSG in 1964. More recent titles have been added for this publication.

and even computer demonstrations. Further information is available, for example, by writing to

Association for Computing Machinery
211 East 43 Street
New York 17, N.Y.

This Guide may also serve school libraries and mathematics clubs in building a collection of digital computer reference materials.

2. DIGITAL COMPUTER TOPICS

This Guide has been written to answer three questions about digital computing and related mathematics:

a. What are the important topics to be studied?

b. In what order should these be studied?

c. Where can information be found on these topics?

Several topics are suggested as basic to an understanding of digital computers. The first five of these, described in the next paragraph, have been intentionally ordered as they are. If this order is followed, a continuity will be developed that should aid the beginner in this study. These five topics are considered fundamental to a thorough study of the field. The final three topics, described later, may be studied in any sequence and are of lesser importance.

The references begin with material on The Nature and Organization of Digital Computers.* The capabilities of computers, the manner in which they are organized, and the means whereby information is stored in them are considered. When preparing a problem for computer solution, it is necessary to formulate an algorithm. This process, termed Problem Analysis, includes stating the problem, selecting a method, analyzing and visualizing it as a step-by-step sequence of operations. In this latter process, emphasis is placed on a flowchart representation, i.e., a diagram displaying the sequence of operations comprising a procedure. The selected references stress this approach. Computer programs are written in various programming languages. Some of these languages consist of statements in a notation similar to mathematical formulas and are termed Algorithmic Languages. References are given to specific languages of this type. The use of appropriate problems for the expression of computer

*Underlined phrases in these paragraphs are used as topical headings in the body of this Guide.

concepts is an essential part of a course in computers. As Additional Sources of Problems, material is selected that contains fully-worked problems for the classroom, showing the relation between algorithms and computer programs. In order to understand the nature of the problems that computers can solve, Mathematics of Computation must be considered. References describe systems of numeration, computer arithmetic, approximations, and methods in arithmetic for the solution of problems too difficult to solve by classical mathematics. These five topics constitute the concepts that should be studied in sequence.

There are many Applications of Computer Systems in a variety of fields. Material here includes readings in such fields as engineering, physics, the behavioral sciences, law, ~~etc., for the enrichment~~ of a teacher's background. Computer Operation, the manner in which a computer operates to solve a problem, is described in the next set of references. Details of the instruction-by-instruction execution of a program and the manner of programming at this level of detail are considered. The final group of materials offers Non-technical and Historical Views of the Computer Field, providing popularized or historical literature. These references do not necessarily probe deeply into particular topics. These three topics are considered of less importance than the five mentioned earlier.

Several topics are excluded from this Guide because they are not directly related to the concepts discussed. These include circuit design, Boolean algebra and circuit components.

3. ORGANIZATION OF THE GUIDE

To aid in the study of each topic, the Guide categorizes suitable references. Each is classified as central, peripheral, or advanced:

a. A central reference is one containing material bearing directly on the topic and embracing the concepts described here. This type of reference is further classified:

A primary central reference is one which is expected to be of greatest value to most high school mathematics teachers.

A secondary central reference is one of less value to teachers.

b. A peripheral reference is one in which the material is specialized or not central to the topic but touches upon it, or is somewhat broader in scope than the topic here defined.

c. An advanced reference contains material central to the topic but which is writte at a higher or more theoretical level.

At the end of the Study Guide, all books are listed alphabetically by author. It is recognized that the list is not exhaustive. Suggestions for appropriate additions are welcomed.

I. NATURE AND ORGANIZATION OF COMPUTERS

The manner in which a computer is organized, with consideration of major elements.

Central (Primary) References

HULL, Introduction to Computing, Ch. 2

LEESON AND DIMITRY, Basic Programming Concepts, Ch. 1, pp. 1-9

SHERMAN, Programming and Coding Digital Computers, Ch. 3, pp. 41-47; pp. 63

SMITH, Computer Programming Concepts, Ch. 1

SPROWLS, Computers--A Programming Problem Approach, Ch. 12

VON NEUMANN, The Computer and the Brain, pp. 1-38

Central (Secondary) References

HULL, Introduction to Computing, Ch. 15

LEESON AND DIMITRY, Basic Programming Concepts, Ch. 2, pp. 10-21

NCTM, Computer Oriented Mathematics, Ch. 2, pp. 19-27

Peripheral References

GREENBERGER (editor), Management and the Computer of the Future, Part 6, pp. 221-248

II. PROBLEM ANALYSIS

Formulation of a method (algorithm) with emphasis on a flow-chart representation.

Central (Primary) References

ARDEN, An Introduction to Digital Computing, Ch. 4, pp. 46-52

GALLER, The Language of Computers

HULL, Introduction to Computing, Ch. 1

NCTM, Computer Oriented Mathematics, pp. 4-18, 59-101, 120-137.

ORGANICK, A FORTRAN Primer, pp. 31-80

ORGANICK, A MAD Primer, pp. 43-87

SHERMAN, Programming and Coding Digital Computers, Ch. 2, pp. 17-40

SMITH, Computer Programming Concepts, Ch. 2

Peripheral References

- BORKO, Computer Applications in the Behavioral Sciences, pp. 114-118
- FROESE, Introduction to Programming the IBM 1620, pp. 15-16
- GREENBERGER (ed.), Management and the Computer of the Future, Part 5, pp. 191-193; p. 211

Advanced References

- ARDEN, An Introduction to Digital Computing
- BRADEN AND PERLIS, An Introductory Course in Computer Programming, pp. 7-34
- TRAKHTENBROT, Algorithms and Automatic Computing Machines

III. ALGORITHMIC LANGUAGE

Introduction to specific languages.

Central (Primary) References

ALGOL

- ANDERSEN, Introduction to ALGOL 60
- BAUMANN et al, Introduction to ALGOL
- MC CRACKEN, A Guide to ALGOL Programming

FORTRAN

- HARRIS, FORTRAN II and IV Programming
- HARVILL, Basic FORTRAN Programming
- HULL, Introduction to Computing, Ch. 4-10
- MC CRACKEN, A Guide to FORTRAN Programming
- ORGANICK, A FORTRAN Primer
- SMITH, Computer Programming Concepts, Ch. 3, pp. 12-16; Ch. 4-9
- SMITH AND JOHNSON, FORTRAN Autotester

OTHERS

- GALLER, The Language of Computers (simplified version of MAD)
- ORGANICK, A MAD Primer
- SPROWLS, Computers--A Programming Problem Approach, Ch. 10, 11 (COBOL), Ch. 14 (PL/I)

Central (Secondary) References

ALGOL

SHERMAN, Programming and Coding Digital Computers, Ch. 14.

FORTRAN

COLMAN AND SMALLWOOD, Computer Language

GERMAIN, Programming the IBM 1620, Ch. 8-9 (GOTRAN)

LEESON AND DIMITRY, Basic Programming Concepts and the IBM 1620 Computer, pp. 174-220

MC CRACKEN AND DORN, Numerical Methods and FORTRAN Programming, Ch. 1, 7, 9, and Appendix 1

SHERMAN, Programming and Coding Digital Computers, Ch. 14

OTHERS

MC GEE, The Formulation of Data Processing Problems for Computers (in ALT, Vol. 4, pp. 3-21) (COBOL)

NCTM, Computer Oriented Mathematics, Ch. 2 (hypothetical language)

Peripheral References

FORTRAN

BORKO, Computer Applications in the Behavioral Sciences, Ch. 7, pp. 124-132

LEESON AND DIMITRY, Basic Programming Concepts and the IBM 1620 Computer, pp. 326-353

Advanced References

ALGOL

DIJKSTRA, A Primer of ALGOL 60 Programming

FORTRAN

MC CRACKEN AND DORN, Numerical Methods and FORTRAN Programming, Ch. 2-6, 8, 10, 11

OTHER

ARDEN, An Introduction to Digital Computing (MAD)

IV. ADDITIONAL SOURCES OF PROBLEMS

Mathematical problems displayed with algorithmic language programs for their solution.

Central (Primary) References

ARDEN, An Introduction to Digital Computing, Ch. 4, pp. 46-52

GALLER, The Language of Computers

JOHNSTON et al, An Introduction to Mathematics, Ch. 1, pp. 46-152

NCTM, Computer Oriented Mathematics, Ch. 3.

ORGANICK, A FORTRAN Primer, pp. 109-155

ORGANICK, A MAD Primer, pp. 181-238

SHERMAN, Programming and Coding Digital Computers

SMITH, Computer Programming Concepts, Ch. 10 (Vol. 1), all of Vol. 2

Central (Secondary) References

GRUENBERGER AND MC CRACKEN, Introduction to Electronic Computers, (good examples scattered throughout book, but done in 1620 machine code)

LARSSON, Equalities and Approximations with FORTRAN Programming, pp. 60-62; p. 104; p. 144

Advanced References

ARDEN, An Introduction to Digital Computing, Ch. 10, pp. 131-147; Ch. 12-18, pp. 161-344

BRADEN AND PERLIS, An Introductory Course in Computer Programming, pp. 81-121

V. MATHEMATICS OF COMPUTATION

Numerical methods; error analysis; approximations; computer arithmetic; systems of numeration.

Central (Primary) References

ARDEN, An Introduction to Digital Computing, Ch. 7-8, pp. 87-112

HARRIS, Numerical Methods Using FORTRAN, Ch. 8-9

KOVACH, Computer-Oriented Mathematics

NCTM, Computer Oriented Mathematics, Appendix A

Central (Secondary) References

BORKO, Computer Applications in the Behavioral Sciences, Ch. 6, pp. 62-11

GOLDEN, FORTRAN IV Programming and Computing

NCTM, Computer Oriented Mathematics, Appendix B

Peripheral Reference

FROESE, Introduction to Programming the IBM 1620, pp. 11-12

Advanced References

ARDEN, An Introduction to Digital Computing, Ch. 7-10, Ch. 12-16

FOX, Introduction to Numerical Linear Algebra

VI. APPLICATIONS OF COMPUTER SYSTEMS

Examples of the use of computers in such fields as engineering, sociology, physics, etc.

Central (Primary) References

BAR-HILLEL, The Present Status of Automatic Translation of Languages (in ALT, Vol. 1, pp. 92-157)

BORKO, Computer Applications in the Behavioral Sciences, Ch. 4, 9, 10, 13, 14, 23, 24

GOTLIEB, General-purpose programming for business applications (in ALT, Vol. 1, pp. 1-42)

GREEN, Digital Computers in Research, Ch. 8-13

LAWLOR, Information technology and the Law (in ALT, Vol. 3, pp. 299-352)

MC GEE, The formulation of data processing problems for computers (in ALT, Vol. 4, pp. 1-52)

SAMUEL, Programming computers to play games (in ALT, Vol. 1, pp. 165-192)

SKRAMSTAD, Combined analog-digital techniques in simulation (in ALT, Vol. 3, pp. 275-298)

Peripheral References

GASS, Recent developments in linear programming (in ALT, Vol. 2, pp. 296-377)

GREENBERGER, Management and the Computer of the Future, Ch. 2, pp. 36-91 (decision making); Ch. 3, pp. 94-130 (simulation of human thinking); Ch. 4, pp. 135-178 (information search and retrieval)

Advanced References

BORKO, Computer Applications in the Behavioral Sciences, Ch. 11, 12, 15 - 22

FEIGENBAUM, Computers and Thought

VII. COMPUTER OPERATION

The manner in which a digital computer operates to solve a problem. Machine-language concepts and programming are included.

Central (Primary) References

BORKO, Computer Applications in the Behavioral Sciences, Ch. 5

DODES, IBM 1620 Programming for Science and Mathematics, Parts II-III, Appendices

FROESE, Introduction to Programming the IBM 1620 (machine language and SPS)

GERMAIN, Programming the IBM 1620

HULL, Introduction to Computing, Ch. 3

LEESON AND DIMITRY, Basic Programming Concepts and the IBM 1620 Computer, Ch. 2-14, pp. 22-173

NCTM, Computer Oriented Mathematics, Appendix A, pp. 138-153

SHERMAN, Programming and Coding Digital Computers, Ch. 3, pp. 47-60

Central (Secondary) References

ARDEN, An Introduction to Digital Computing, Ch. 1, 5, 6

MC CORMICK, Digital Computer Primer (hypothetical computer)

NCTM, Computer Oriented Mathematics, Ch. 2, pp. 27-40 (hypothetical computer)

SMITH, Computer Programming Concepts, Ch. 3

Peripheral References

CODD, Multiprogramming (in ALT, Vol. 3, pp. 78-153)

CURTIN, Multiple computer operations (in ALT, Vol. 4, pp. 245-303)

ENGLEBART, Games that teach the fundamentals of computer operation

GREEN, Digital Computers in Research, Ch. 15

GRUENBERGER AND MC CRACKEN, Introduction to Electronic Computers

Advanced References

MC NAUGHTON, The theory of automata, a survey (in ALT, Vol. 2, pp. 379-421)

VIII. NON-TECHNICAL AND HISTORICAL VIEWS OF THE COMPUTER FIELD

Popularized or historical literature in the field.

Central (Primary) References

BERNSTEIN, The Analytical Engine: Computers--Past, Present and Future

BORKO, Computer Applications in the Behavioral Sciences, Ch. 3 (history)

NCTM, Computer Oriented Mathematics, Appendix E, pp. 198-200 (short history)

TOMPKINS, Computer education (in ALT, Vol. 4, pp. 135-168)

VON NEUMANN, The Computer and the Brain, Part 2, pp. 39-82 (the human nervous system)

Central (Secondary) References

DARNOWSKI, Computers--Theory and Uses, Vol. 1, pp. 1-29, 61-70

Peripheral References

GREENBERGER, Management and the Computer of the Future, Ch. 1, pp. 2-34; Ch. 8, pp. 291-324

Advanced References

SHOULDERS, Micro-electronics using electron-beam-activated machining techniques (in ALT, Vol. 2, pp. 137-293)

ANNOTATED BIBLIOGRAPHY

ALT, F. L. (editor); Advances in Computers, Vols. 1-4. New York: Academic Press, 1960-1963. The articles in these books range from introductions for certain fields and summaries of existing work in a particular field to quite technical papers to be read only with an appropriate background. The following selection of twelve of these articles has been chosen as particularly suitable for the high school teacher.

BARR-HILLEL, Y.; The present status of automatic translation of languages (Vol. 1, pp. 92-157). A survey of the field of translation of natural languages, e.g., French, describing the accomplishments of a number of workers. Some of the problems encountered in natural language translation are described in an appendix.

GOTLIEB, C. C., General-purpose programming for business applications (Vol. 1, pp. 1-42). A rapid, extensive overview of the business data processing field containing a short introduction to programming, systems; characteristics of data processing problems; typical data processing operations (sorting, merging, file handling).

SAMUEL, A. L., Programming computers to play games (Vol. 1, pp. 165-192). Written in general terms without giving details of algorithms, this article deals primarily with computer checker-playing, although it also discusses several approaches to computer chess-playing.

GASS, S. I., Recent developments in linear programming (Vol. 2, pp. 296-377). A survey of the field of linear programming (minimization of a linear function of several variables over a region defined by boundaries specified by linear equations). Several specific programming languages and some applications are described.

MC NAUGHTON, R., The theory of automata, a survey (Vol. 2, pp. 379-421). Automata theory (excluding such topics as switching theory, theory of computability, and artificial intelligence) is treated starting from a basic level, defining fundamental concepts, and proceeding at a level appropriate for most teachers.

SHOULDERS, K. R., Micro-electronics using electron-beam-activated machining techniques (Vol. 2, pp. 137-293). A lengthy (150-page) report on devices that may have considerable impact on computer design technology. Thin film circuitry is described in great detail with emphasis on novel manufacturing techniques.

CODD, E. F., Multiprogramming (Vol. 3, pp. 78-153). Multiprogramming is concerned with concurrency of operations within computer systems. This article offers a thorough, relatively non-technical introduction to the subject. It requires no particular background in a specific programming language.

LAWLOR, R. C., Information technology and the law (Vol. 3, pp. 299-352). This article provides a brief look at the possible utilization of computers to aid in information retrieval in the field of law. In addition, the article shows how Supreme Court decisions might be predicted by the computer, using information on past decisions.

SKRAMSTAD, H. K., Combined analog-digital techniques in simulation (Vol. 3, pp. 275-298). The kinds of problems best handled on such a system are described; some equations are solved on each of the two types. Examples are given. The problems with such a system are described.

CURTIN, W. A., Multiple computer operations (Vol. 4, pp. 245-303). A description of general concepts for the design, programming, and scheduling of multiple computer systems. Existing multiple computer systems are reviewed.

MC GEE, W. C., The formulation of data processing problems for computers (Vol. 4, pp. 1-52). A review of recent developments in certain areas of data processing: the characteristics of data processing languages (COBOL et al), organization and description of data, and some beginning attempts at a theory of data processing.

TOMPKINS, H. E., Computer education (Vol. 4, pp. 135-168). A description of the efforts toward computer education at the college and high school levels with emphasis on the former. A few comments are given on programmed instruction.

ANDERSEN, C. An Introduction to ALGOL 60. Reading, Mass.: Addison-Wesley, 1964. A clear and very brief description of the language ALGOL. The reader is assumed to have a basic knowledge of step-by-step logical processes and repetitive operations. The features of the language are introduced gradually and in a natural and convenient order. Individual features are illustrated well by examples.

ARDEN, B. W. An Introduction to Digital Computing. Reading, Mass.: Addison-Wesley, 1963. This book for the scientifically-minded reader is an excellent introduction to digital computing. About one-half of the book is devoted to a detailed exposition of the subject of numerical analysis. Many numerical techniques are illustrated by algorithms expressed in the MAD language. The book also contains an excellent chapter on numerical methods and a final chapter which describes the programming of "a simple compiler". In addition to an introduction to MAD, a basic approach to machine organization is given.

BAUMANN, FELICIANO, BAUER AND SAMELSON. Introduction to ALGOL. Englewood Cliffs, N. J. Prentice-Hall, Inc., 1964. An excellent, exceptionally clear and concise textbook on the language ALGOL; well-suited for use as a reference work as well (includes revised report on ALGOL 60 as appendix). Assumes a knowledge of the basic ideas of step-by-step logical procedures and repetitive processes.

BERNSTEIN, J. The Analytical Engine: Computers--Past, Present and Future. New York: Random House, 1964. This highly readable little book (103 pages) first appeared as a series of articles in the New Yorker. Any reasonably literate person could enjoy it as popular historical background.

BORKO, H. (editor) Computer Applications in the Behavioral Sciences. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1962. Although comprised principally of a collection of 17 reports covering a wide variety of computer applications to the behavioral sciences, this book also contains a 140-page introduction to computing which presupposes no prior knowledge. The research reports, while not easy reading, should be rewarding for the teacher who wishes to delve further into any of the topics.

BRADEN AND PERLIS. An Introductory Course in Computer Programming. This is Monograph No. 7 of the Discrete Systems Concepts Projects funded by the National Science Foundation and written at the Carnegie Institute of Technology, Pittsburgh, Penna. These course notes, not commercially distributed, are for an introductory course primarily involved with ALGOL. They are well written and contain many fine exercises as well as an enlightening exposition of data structures.

COLMAN AND SMALLWOOD. Computer Language--An Auto-instructional Introduction to FORTRAN. New York: McGraw-Hill, 1962. An introductory manual (in a rather unusual format) to a subset of the language FORTRAN.

DARNOWSKI, V. S. Computers--Theory and Uses (teaching unit and teachers' guide). Washington, D. C.: National Science Teachers Association, 1964. Limited editions--revised edition to be offered for sale at a later date.

DIJKSTRA, E. W. A Primer of ALGOL 60 Programming. New York: Academic Press, 1962. A brief, well-written readable presentation of ALGOL 60 to readers already familiar with some compiler language. Special features of the implementation of the language for the Mathematical Centre, Amsterdam, are presented.

DODES, I. A. IBM 1620 Programming for Science and Mathematics. New York: Hayden Book Co., 1963. Of interest primarily to those who have access to a 1620 computer, this is a text in 1620 programming for good 12th grade students. It is not an introduction to the overall field of computing, but treats numerical analysis, machine language and symbolic programming, and--briefly--FORTRAN.

ENGLEBART, D. C. Games that teach the fundamentals of computer operation, IRE Transactions, Vol. E.C.-10, No. 1, March 1960. This paper instructs a teacher in the rules for playing a game using up to 20 students for simulating various kinds of simple computer elements. Each individual watches the up-down hand position of one or two others and adjusts his hand position to a response task which is equivalent to an AND, OR, NOT, or flip-flop. Counters, shift registers, and adders may be organized in this way.

FEIGENBAUM AND FELDMAN Computers and Thought. New York: McGraw-Hill, 1963. A fine collection of twenty research reports on Artificial Intelligence (programming computers to perform intellectual tasks such as game-playing, theorem-proving, etc., in the same way that persons might perform these tasks) and Simulation of Cognitive Processes (construction of computer models to aid in understanding the information processes underlying human behavior).

FOX, L. Introduction to Numerical Linear Algebra. Oxford: Oxford University Press, 1964. A sound, readable account, mostly at the level of intermediate algebra, of the numerical methods used in the solution of linear equations, matrix inversion, and the eigenvalue problem.

FROESE, C. Introduction to Programming the IBM 1620. Reading, Mass.: Addison-Wesley, 1964. Of interest only for those who have access to an IBM 1620 computer. Emphasis on operation, feeding information into the machine, machine language and symbolic programming.

GALLER, B. A. *The Language of Computers*. New York: McGraw-Hill, 1962. An excellent introduction to the structure and use of a machine-independent algorithmic language (a simplified version of MAD). The language is introduced gradually, employing examples together with solutions given both by flow-chart and MAD program. A complete answer book is available on request from the publisher.

GERMAIN, C. B. *Programming the IBM 1620*. Englewood Cliffs, N. J.: Prentice-Hall, 1962. A text for a first course in programming with emphasis upon the operation of the 1620 (assumed to be available) and the use of machine language and symbolic coding.

GOLDEN, J. T. *FORTRAN IV Programming and Computing*. Englewood Cliffs, N. J.: Prentice-Hall, 1965. A development of the FORTRAN IV language. It differs from some of the others in its heavier use of mathematical examples. The mathematics is at the early college level, and should give teachers some ideas for examples and exercises.

GREEN, B. F., Jr. *Digital Computers in Research*. New York: McGraw-Hill, 1963. This book is of interest principally for the applications in Part III. Many of these problems in the behavioral sciences are treated rather lightly and can be profitably read without attention to Parts I and II.

GREENBERGER, M. (editor) *Management and the Computer of the Future*. Cambridge, Mass.: M.I.T. Press, 1962. In spite of its misleading title, this collection of eight lectures (with accompanying discussion) contains much background material of special interest to high school teachers and students. (Not a textbook.)

GRUENBERGER AND MC CRACKEN. *Introduction to Electronic Computers*. New York: John Wiley, 1963. A good IBM 1620 machine-language programming text suitable for 12th grade students having access to a 1620 computer. A very good presentation of basic and important (but highly machine-oriented) material.

HARRIS, L. D. *FORTRAN II and IV Programming*. Columbus, Ohio: Charles E. Merrill, 1964. This book contains a brief introduction to FORTRAN. Emphasis is on a simple subset of FORTRAN. This material is reprinted in the same author's text, *Numerical Methods Using FORTRAN*.

HARRIS, L. D. *Numerical Methods Using FORTRAN*. Columbus, Ohio: Charles E. Merrill, 1964. This book attempts a marriage of programming and numerical methods for the engineer or scientist. Although the presentation of FORTRAN is quite readable, the book is primarily of interest for the problems in Chapters 8 and 9.

HARVILL, J. B. *Basic FORTRAN Programming*. Englewood Cliffs, N. J.: Prentice-Hall, 1966. An interesting introduction to a small version of FORTRAN (essentially FORTRAN II without format details). Written in an elementary style and emphasizing the usefulness of flow diagrams throughout, this book should be attractive to students. The author introduces an interesting technique for watching the changes of pertinent storage areas over time, called the Memory Chart.

HULL, T. E. *Introduction to Computing*. Englewood Cliffs, N. J.: Prentice-Hall, 1966. A readable introduction to computing via FORTRAN IV, with several non-numerical examples in the later chapters. The bibliography in Appendix A is very good.

JOHNSTON, PRICE AND VAN VLECK. An Introduction to Mathematics, Vol. 1, Parts 1 and 2. Lawrence, Kansas; Department of Mathematics, The University of Kansas, 1963. This book is part of a mathematics text for the University freshman. Only pages 46-152 are of interest to this study guide. In that segment of the book the basic concepts of flow charting and programming an algorithm (in an informal language similar to ALGOL) are presented in the context of solving systems of linear equations: 2 × 2, 4 × 5, and m × n. This section is recommended as an example of a quite detailed algorithmic solution of a problem, not as a text.

KOVACH, L. D. Computer-Oriented Mathematics. San Francisco: Holden-Day, 1964. A small book, easy to read. It is concerned with such computer-oriented topics as approximations, iteration, Monte Carlo methods, etc. There is no computer programming here, just the mathematics.

LARSSON, R. D. Equalities and Approximations: With FORTRAN Programming. New York: John Wiley and Sons, Inc., 1963. The teacher who knows FORTRAN can find in this book a few problems in mathematics programmed in FORTRAN (basic formatless FORTRAN for the IBM 1620).

LEESON AND DIMITRY. Basic Programming Concepts and the IBM 1620 Computer. New York: Holt Rinehart and Winston, 1962. This text is a complete treatment of programming the 1620 with detailed emphasis on machine language and symbolic coding. A brief (probably too brief for use by a beginner) but accurate presentation of 1620 FORTRAN I is given.

MC CORMICK, E. M. Digital Computer Primer. New York: McGraw-Hill Book Co., 1959. The person already competent in the area of programming and who wishes to delve into how computers work internally and how they are designed can find in this book a brief treatment of arithmetic and logical units, input-output devices and related topics.

MC CRACKEN, D. D. A Guide to ALGOL Programming. New York: John Wiley and Sons, Inc., 1962. This is a well-organized introduction to the language ALGOL, including nine case-study examples, each carrying a problem from the original statement through the completed ALGOL program.

MC CRACKEN, D. D. A Guide to FORTRAN Programming. New York: John Wiley, 1961. A brief (38 pages) introduction to FORTRAN programming for the person who wants to get a rapid grasp of the language.

MC CRACKEN AND DORN. Numerical Methods and FORTRAN Programming. New York: John Wiley, 1964. A very readable book providing an adequate description of FORTRAN and a good introduction to a well-selected set of topics in numerical analysis. Aimed at under-graduates in science and engineering, many parts of the book are likely to be too advanced to suit the needs of the high school teacher. However, the teacher with adequate mathematical background will find much of the material useful for his own enrichment even though most of it will be beyond the reach of his students.

NATIONAL COUNCIL OF TEACHERS OF MATHEMATICS. Computer Oriented Mathematics, An Introduction for Teachers. Washington, D. C.; N.C.T.M., 1963. This book has an excellent plan as an introduction for teachers. Its purpose is not to teach the idea of a computer as an end in itself, but rather to motivate the study of mathematics by drawing upon the appeal and power of computers. In order to attain this goal, certain problems of mathematics are selected which can be solved appropriately on a computer. Emphasis is placed on the organization of solutions into logical step-by-step processes, the use of flow-charts, and on the repetitive capabilities of computers.

ORGANICK, E. I. A FORTRAN Primer. Reading, Mass.: Addison-Wesley, 1963. This is one of the most complete and well-organized FORTRAN texts available. It uses a comprehensive set of examples and drill exercises independent of any discipline. Its completeness and thoroughness in treating the differences between FORTRAN processors on various machines may make this book more suitable for a course taught by a teacher with previous knowledge of FORTRAN than for self-study.

ORGANICK, E. I. A MAD Primer. Ann Arbor, Mich.: Ulrich's Book Store, 1964. This book does for the MAD language precisely what the author's A FORTRAN Primer does for FORTRAN.

SHERMAN, P. M. Programming and Coding for Digital Computers. New York: John Wiley, 1962. An excellent comprehensive source book of information on basic computer concepts and on computer programming, including numerical scientific applications, business data processing, and non-numerical applications. Probably more useful to the high school teacher as a reference work than as a text.

SMITH, R. E. Computer Programming Concepts. Vol. 1 (Reference material), Vol. 2 (Problem exercises). Minneapolis, Minn.; Control Data Corp., 1963. An excellent introduction to the basic concepts of computers at a level easily understood by high school students. Emphasis is on FORTRAN as used with the Control Data 160-A Computer. Well-selected examples with a liberal sprinkling of humor.

SMITH AND JOHNSON. FORTRAN Autotester. New York; John Wiley, 1961. An excellent introduction to FORTRAN, particularly if a computer is not available for program check-out during the course of study. Minimal use of flowcharts. Exceptionally well-suited to a brief self-instructional initiation to the FORTRAN Language.

SPROWLS, R. C. Computers--A Programming Problem Approach. New York: Harper and Row, 1966. A good treatment of several languages, such as FORTRAN, COBOL, and PL/I. Many examples, with emphasis on flow diagrams. The accompanying instructor's manual contains a great deal of teaching philosophy and insight into the author's teaching methods. This book will be useful primarily for the teacher.

TRAKHTENBROT, B. A. Algorithms and Automatic Computing Machines. (Translated from the Russian edition--1960). Boston: D. C. Heath and Co., 1963. This book is concerned with the theory of algorithms. It requires no specific information from other branches of mathematics beyond intermediate algebra. The subject matter is deep and the treatment is rigorous, requiring the reader to follow a rather complex train of logical thought, but the author has done an excellent job of making the ideas as accessible as possible. The basic ideas are introduced very carefully and gradually, and they are very well motivated. Recommended for the teacher who would like to follow up some of the logical and philosophical implications of computing.

VON NEUMANN, J. The Computer and the Brain. New Haven: Yale University Press, 1958. This excellent book, although not intended as a textbook, is recommended to the teacher as an historically oriented account of the organization of computing machines. The second part of the book discusses analogous properties of the human nervous system.